IST-2001-32133

GridLab – A Grid Application Toolkit and Tesbted

D9.2 “Technical Specification Document”

Author(s):	Jerzy Brzezinski, Jarek Nabrzyski, Juliusz Pukacki, Tomasz Piontek, Krzysztof Kurowski, Bogdan Ludwiczak, Radoslaw Strugalski, Maciej Hapke, Nikolaos Doulamis, Anastasios Doulamis, Manos Varvarigos, Konstantinos Dolkas.
Title:	Technical Specification Document
Subtitle:	GridLab Resource Management System
Work Package:	9
Lead Partner:	PSNC
Partners:	NTUA, AEI, TRIANA
Filename:	GridLab-9-D.2-0001-Technical Specification Document
Version:	1
Config ID:	GridLab-9-D.2-0001-1.0
Classification:	IST

Abstract: This document presents the GridLab Resource Management System, its architecture and design. This document will be continuosly updated.

Project Manager: Jaroslaw Nabrzyski
Institute of Bioorganic Chemistry PAS
Poznan Supercomputing and Networking Center
ul. Noskowskiego 12/14
61-704 Poznan, Poland
Phone:+48 61 858 2072, Fax: +48 61 852 5954
Email: naber@man.poznan.pl

1. Introduction

It is clear that sophisticated, fault tolerant superscheduling mechanisms based upon reservation facilities and performance prediction are required in order to efficiently utilize Grid environments. But much more than this, in a dynamic environment, the needs of the processes running on the Grid will change, resources will change, and so a mechanism must exist to monitor the present needs, the expected performance, and the actual performance, and to adapt accordingly. Such capabilities do not exist today, but will be critical for the types of applications and Grids we envision.

Within the GridLab Project we are developing GridLab Resource Management System (GRMS) which will cope with all the mentioned problems. We focus on efficient and effective use of resources by mapping job resource requests in a way that will satisfy both the application users and resource administrators. The GRMS will help to meet applications, users and administrators requirements based on user's preferences, multi-criteria resource performance evaluation and prediction-based scheduling. Moreover, our system is trying to look into the system-specific and job-specific features, such as schedulability, resource availability and utilization, data-access communication delays, history of previous job execution and queue wait times as well as data access and communication time. All these parameters are taking into account and allow us to use various scheduling algorithms in the GRMS, including min CT, backfilling, as well as some other heuristics. Thus, the GRMS, based on dynamic resource selection and advanced scheduling methodology, combined with feedback control architecture, deals with dynamic Grid environment and resource management challenges and appears like a new essential approach to the complex Grid related problems and experiments.

All the functionality of the GRMS is straightforward and transparent for end-users and simultaneously available for third party resource managers or specific applications through the set of well defined interfaces. We assume that API's of the GRMS will be also available to the grid application developer as a part of Grid Application Toolkit (GAT). Up to date, we designed and developed basic components of the GRMS to facilitate job and task submissions, monitoring and application steering.

This document is a Grid[Lab] Resource Management Software (GRMS) Architecture Document and it provides an overview of the entire software architecture. Section 1 gives an overview of the requirements and overall architecture. Section 2 gives some possible resource management scenarios in the GridLab project, including the migration scenario. Further sections present the core section of the software architecture document. Appendix 1 presents the scheduling algorithms proposed by GridLab, the research on which will be performed using the GRMS software components.

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

Software Architecture Document specifies main components and services of the GridLab Resource Management System, such as job description language, broker, resource discovery service, job id management service, resource evaluation service, planner and scheduler etc. The architecture presented here affects and influences (and on the other hand is influenced by) all the work packages of the GridLab project. It is also shaped by the requirements of the GridLab and all the other applications the authors of this document talked to.

1.3 Definitions, Acronyms, and Abbreviations

The following definitions, acronyms and abbreviations are used in this document:

Application: Application is a general term describing entity which utilizes functionality provided by GridLab Resource Management System. It is the only actor that initiates all use cases for GRMS. From the implementation point of view an Application is a implementation of GAT API. As the Application also portal is considered.

Interoperability - is the ability of a system or a product to work with other systems or products without special effort on the part of the customer. Interoperability becomes a quality of increasing importance for information technology products as the concept that "The network is the computer" becomes a reality. For this reason, the term is widely used in product marketing descriptions.
Products achieve interoperability with other products using either or both of two approaches:

- By adhering to published interface standards
- By making use of a "broker" of services that can convert one product's interface into another product's interface "on the fly"

A good example of the first approach is the set of standards that have been developed for the World Wide Web. These standards include TCP/IP, Hypertext Transfer Protocol, and HTML. The second kind of interoperability approach is exemplified by the Common Object Request Broker Architecture (CORBA) and its Object Request Broker (ORB).

Architecture
In information technology, especially computers and more recently networks, architecture is a term applied to both the process and the outcome of thinking out and specifying the

overall structure, logical components, and the logical interrelationships of a computer, its operating system, a network, or other conception. An architecture can be a *reference model*, such as the Open Systems Interconnection (OSI) reference model, intended as a model for specific product architectures or it can be a specific product architecture, such as that for an Intel Pentium microprocessor or for IBM's OS/390 operating system.
Computer architecture can be divided into five fundamental components: input/output, storage, communication, control, and processing. In practice, each of these components (sometimes called *subsystems*) is sometimes said to have architecture, so, as usual, context contributes to usage and meaning.
By comparison, the term *design* connotes thinking that has less scope than architecture. Architecture is a design, but most designs are not architectures. A single component or a new function has a design that has to fit within the overall architecture.

Scalability
In information technology, scalability seems to have two usages:

- It is the ability of a computer application or product (hardware or software) to continue to function well as it (or its context) is changed in size or volume in order to meet a user need. Typically, the rescaling is to a larger size or volume. The rescaling can be of the product itself (for example, a line of computer systems of different sizes in terms of storage, RAM, and so forth) or in the scalable object's movement to a new context (for example, a new operating system).
- It is the ability not only to function well in the rescaled situation, but to actually take full advantage of it. For example, an application program would be scalable if it could be moved from a smaller to a larger operating system and take full advantage of the larger operating system in terms of performance (user response time and so forth) and the larger number of users that could be handled.

It is usually easier to have scalability upward rather than downward since developers often must make full use of a system's resources (for example, the amount of disk storage available) when an application is initially coded. Scaling a product downward may mean having to achieve the same results in a more constrained environment.

Interface
As a noun, an interface is either:

- A user interface, consisting of the set of dials, knobs, operating system commands, graphical display formats, and other devices provided by a computer or a program to allow the user to communicate and use the computer or program. A graphical user interface (GUI) provides its user a more or less "picture-oriented" way to interact with technology. A GUI is usually a more satisfying or user-friendly interface to a computer system.
- A programming interface, consisting of the set of statements, functions, options, and other ways of expressing program instructions and data provided by a program or language for a programmer to use.

- The physical and logical arrangement supporting the attachment of any device to a connector or to another device.

As a verb, to interface means to communicate with another person or object. With hardware equipment, to interface means making an appropriate physical connection so that two pieces of equipment can communicate or work together effectively.

API

An Application Program Interface (API - and sometimes spelled *application programming interface*) is the specific method prescribed by a computer operating system or by an application program by which a programmer writing an application program can make requests of the operating system or another application.

Service

A service is a network-enabled entity that provides a specific capability. [...] A service is defined in terms of the protocol one uses to interact with it and the behavior expected in response to various protocol message exchanges (i.e., service = protocol + behavior) (from "Physiology of the Grid"). Within GridLab however the following service categories are also being considered:

Web Service: The term Web services describes an important emerging distributed computing paradigm [with] focus on simple, Internet-based standards (e.g., eXtensible Markup Language: XML [...]) to address heterogeneous distributed computing. Web services define a technique for describing software components to be accessed, methods for accessing these components, and discovery methods that enable the identification of relevant service providers.

Grid Service: A Web service that provides a set of well-defined interfaces and that follows specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability; the conventions address naming and upgradeability.

OGSA Service: This is a Grid Service compliant with an Open Grid Service Architecture specification.

GridLab Service: A service provided by the GridLab project, normally a Grid Service. All GRMS services will be Grid Services in general and in a short future (March-May, 2003) they will be OGSA services. The following abbreviations are used to refer to GRMS services:

JR – Job Receiver
RDS – Resource Discovery Service
RES – Resource Evaluation Service
ACS – Adaptive Component Service (developed by WP7)
BS – Brokering Service
DWS – Distributed Workflow Service

PS – Prediction Service
LTS – Logging and Tracking Service
ARS – Advanced Reservation Service
QoS – QoS Negotiation Service
REST – Resource Estimation Service

Third-party Service: A service provided outside the scope of GridLab, either from underlying Grid middleware or from legacy software.

Java Program: Business process applications written in Java. The requested job will run as an EJB component in an EJB container hosting environment or as a Java Servlet in a Servlet Container. Most newly written applications are expected to be of this type. Job requests of this type can be either static or dynamic. Job lifetimes may be very different.

Batch Process: Current business process applications also include batch type jobs including periodic ones, e.g., monthly summary report creation or employee payment transfers. In some cases, workflow management is also required. Requirements may be the same as for job execution using the GRAM interface of Globus Toolkit v2.x. Most job requests of this type are dynamic and have relatively short lifetimes.

System Synthesis: Terraspring provides an innovative feature to create and deploy entire business process applications using multiple servers. We call this feature "system synthesis." IDC administrators may submit this kind of job request. Such requests are static, very infrequent, and are expected to have very long lifetimes.

Fault tolerance: Fault-tolerant describes a computer system or component designed so that, in the event that a component fails, a backup component or procedure can immediately take its place with no loss of service. Fault tolerance can be provided with software, or embedded in hardware, or provided by some combination.

2. GridLab Application Scenarios and Requirements towards GRMS

The computing requirements for many of today's scientific applications have far outgrown the resources most any one scientific institution is able to provide. These requirements can include the need for more computational power (number and type of processors, physical memory), including availability (time), more communication resources (bandwidth, rate of transfer, reliability), or access to large-scale data resources (mass secondary and tertiary storage). Furthermore, many applications, including the Particle Physics Data Grid described below, by definition require access to distributed data and computational services.

In order to solve these problems, scientists can employ Grid technologies in their applications to exploit the Grid infrastructures that link institutions and enable access to the computing systems those institutions make available for academic-wide use. However, this leads to still other problems. From the application developer's perspective, these include how to effectively locate resources in distributed environments, determine their worth and availability to applications, and then to acquire the use of resources, perhaps at a given cost. Additionally, scientists require mechanisms for monitoring the progress of their applications in these distributed environments, as application processes and data may migrate among multiple computing resources.

Scientists would rather not have to worry about these kinds of problems. They would rather focus on developing applications and not on developing any systems that may be required to support their applications in Grid environments. Instead, application developers ought to be able to rely upon the use of Grid resource brokers, or services that negotiate the use of resources wherever possible on behalf of applications. At the same time, however, it is critical for developers of Grid resource brokers to be aware of the kinds of computing challenges that face developers of scientific applications. Thus, in this section we provide real-world examples of Grid application projects, projects that intend to use Grid technologies to support their application resource requirements, and describe how these projects impact the development of Grid resource brokers.

2.1 Cactus Toolkit Requirements

The Cactus Toolkit is a modular simulation code framework developed primarily by the Numerical Relativity Group at the Max Planck Institute for Gravitational Physics. Cactus consists of a "flesh" which coordinates the activities of modules, referred to as "thorns". Cactus thorns interoperate via standard interfaces and communicate directly with the flesh (not with each other). This makes the development of thorns independent of one another, which makes it possible for coordinating geographically dispersed research

groups, and furthermore, allows each group to concentrate on its own area of expertise. For example, physicists may focus on the development of physics thorns (e.g. black hole or hydro evolutions), while computer scientists may develop thorns that support, for instance, interoperability with Grid services (e.g. GRAM-based job schedulers, MDS-based information services).

Cactus has been successfully applied to a wide range of applications, including the simulation of complex astrophysical phenomenon, as in the merger of neutron stars or collisions of black holes. Such applications are both computationally expensive and can, in some cases, require the simultaneous use of multiple supercomputing systems during execution. To support such high-end parallelism, a Cactus thorn has been developed to enable Cactus applications to use MPICH-G, an implementation of MPICH by the Globus team that supports MPI over TCP links.

However, in order to enable Cactus applications to acquire the use of large-scale distributed computing resources, developers of the Cactus Toolkit have been working with several Grid resource broker projects, both to communicate their application needs and to determine new ways in which Cactus can be designed to exploit Grid environments. For instance, the Cactus team designed and built a Cactus application server that enables Cactus applications to migrate among multiple resources during the course of execution. Scenarios that would support the need for migration include cases in which computing resources fail to meet basic application requirements (Contract violation), as well as cases in which a simulation requires more computing time than was initially allotted to the simulation. Cactus application migration provides a rich set of problems to which Grid resource brokers can be applied. Because many of these ideas may be exported to other computing projects, the GrADS team and the Grid Resource Broker group at PSNC have been working with the Cactus team to prototype general migration solutions.

2.2 GridLab general scenarios (questions and requirements)

In this section we describe different scenarios for the GridLab applications. The scenarios include some questions and requirements which will be answered throughout the process of designing the GRMS system.
The end technology developed through this project will enable scenarios, such as the following hypothetical examples, to become reality.

I. Gravitational Wave Detection and Analysis: The gravitational wave detector network, including GEO600 in Germany, collects a TByte of data each day, which must be searched using different algorithms for possible events such as black hole or neutron star collisions, or pulsar signals.
Routine realtime analysis of gravitational wave data from the Hannover detector identifies a burst event, but this standard analysis reveals no information about the burst location. To obtain the location, desperately required by astrophysicists for turning their

telescopes to view the event before it fades, a large series of templates must be cross-correlated against the detector data. An Italian astrophysicist accesses the ***GEO600 Portal****, and using the* ***performance*** *tool finds that 3 TFlops/s is needed to analyze the 100GB of raw data in the required hour. Local resources are insufficient, so using the* ***brokering*** *tool, she locates the fastest available machines around the world. She selects five suitable machines, and with* ***scheduling*** *and* ***data management*** *tools, data is moved, executables created and the analysis starts. In an Amsterdam bar twenty minutes later, an SMS message from the portal's* ***notification*** *tool, informs her that one machine is overloaded, breaking the* ***runtime contract****. She connects with her PDA to the portal, and instructs the* ***migration*** *tool to move this part of the analysis to a different machine. Within the specified hour, a second SMS message tells her analysis is finished, and the resulting data is now on her local machine. Using this location data, observatories are able to find and view an exceptionally strong gamma-ray burst, characteristic of a collision of neutron stars.*

II. Numerical Relativity: A single simulation of an astrophysical event, *e.g.* black hole or neutron star collisions, ideally requires over TByte and TFlop resources, not yet available on a single machine.

Learning more about the detected burst requires cross-correlating detector data with custom wave templates from full-scale neutron star simulations. Sufficiently accurate templates require running large scale simulations too big to fit on any current supercomputer. German members of international numerical relativity collaboration are tasked with creating collision templates for ten different neutron star mass combinations. They access the web-based ***Simulation Portal****, selecting required code modules and building parameter files with the* ***code composition*** *tool. The* ***performance prediction*** *tool estimates that each simulation requires 1024GB of memory and* 10^{14} *Flops, with an additional* 50^{14} *Flops required for processing data to create signal templates. The* ***brokering*** *tool finds that no single machine in the* ***Simulation Testbed*** *can supply enough memory, but locates two machines which can be connected to form a large enough virtual supercomputer, the dynamic grid* ***monitoring*** *tool indicates an acceptable bandwidth between them. The* ***scheduling*** *tool stages the five runs to appropriate queues on the machines, and the first simulation starts. The time-consuming task of creating templates is handled by* ***spawning simulations*** *to smaller machines dynamically located by the* ***broker****, at each time step data is streamed to a series of networked computers for analysis, creating a* ***simulation vector*** *using available machines on the grid. Collaborators around the world connect to the portal using networked workstations, home PCs and modems, as well as the latest wireless PDAs and mobile phones. They are able to use various* ***remote access*** *tools to* ***visualize*** *data,* ***monitor*** *performance and simulation properties, and* ***interactively steer*** *the simulation.*

Hereafter we present some simple scenarios/requirements obtained from the GridLab application teams.

2.2.1 Job submission scenario

We begin with a simple sounding scenario: starting a job on some resource. This is a basic Grid operation that will be used in many more complex scenarios, and as we show it has several variations that already require complex interactions between modules developed by different WPs.

Simple Start Job

This is simplest possible scenario: a known job needs to be started somewhere, asap. The request to start the job may come from any source, such as a user, a portal, a simulation, etc, which we do not deal with right now. Symbolically, the scenario looks like:

```
request ---> resource broker
resource broker ---> |I| find resource set
                     |N| stage application prerequisites on resource set
                     |F| start application
                     |O|
```

Let us analyze this in more detail. It is actually a sequence of steps. First we list the basic sequence of events, and then repeat this with a number of optional events. Comments and questions are listed in italics.

1. A resource request is formulated, and a GAT call is made to find the resource. This call insulates the user from technical details, different resource brokers (RBs), OGSA, Class Ads, etc. The call is made either from the application, through a GAT call, or by the Portal, but again, through a GAT call. The request consists of knowledge of an executable's location, or ability to create one, a parameter file, possibly some input data set, and an understanding of the computational requirement.

2. The request is sent to an RB, through an appropriate adaptor, which evaluates the request and returns an answer. For now, we assume the request included a command to start the job, but it need not.
COMMENTS: If the RB cannot be located, or does not respond in some period of time, localhost could be a default.

3. The RB starts the job on the chosen machine.
Now, repeat the same steps, but with a number of optional items given below, to show the complexity and choices that can be made for even the simplest scenario. Also, details can be debated, but these are the kinds of things that have to be done!

1. Optional: Resource needs have to be estimated. This can be provided by a user, or by the application itself, or by an external resource estimator service.

2. Optional: The request to find the needed resources is initiated from a Portal.

3. A resource request is formulated, and a GAT call is made to find the resource. This call insulates the user from technical details, different resource brokers (RBs), OGSA, Class Ads, etc. The call is made either from the application, through a GAT call, or by the Portal, but again, through a GAT call. The request consists of knowledge of an executable's location, or ability to create one, a parameter file, possibly some input data set, and an understanding of the computational requirement.

COMMENTS: The GAT Engine adaptors translate and transmit this request to the appropriate medium, whether it is OGSA or something else. Hence, we can use OGSA and non-OGSA services. In particular, we can build something NOW, that works, and as OGSA is developed, we will easily incorporate it.

4. The request is sent to an RB, through an appropriate adaptor, which evaluates the request and returns an answer. For now, we assume the request included a command to start the job, but it need not.
COMMENTS: If the RB cannot be located, or does not respond in some period of time, localhost could be a default.

5. Optional: The RB could check a GIS for information, which can tell it which resources are available. The monitoring package could have registered with the GIS, information such as network bandwidth between sites, which may be part of the request.

6. The RB starts the job on the chosen machine. Note that this could mean the job started interactively, was successfully submitted to an interactive queue, or was successfully submitted to a queue to be started at some time in the future.

7. Optional: Although in this example, we assume an executable already exists on the target machine, in principle the executable, input file, and input data will be staged to the resource and executed.
QUESTIONS: How is this done? The RB can take care of it, as above, or alternatively the application or even Portal could use GAT move file calls to stage the executable and associated files, and then make the request to execute the job.

8. Optional: The running executable, once started, registers itself with the application manager or portal.
QUESTIONS: How does this happen? What exactly does the new simulation register with? Where does the job get its unique ID from? From the RB? From itself ? Is the ID given as the return call from announcing call? Again, is UDDI/WSDL used in any way here? What happens if it cannot register, due to firewall, some failure, etc?

This is a basic scenario that brings many elements into one simple grid action. Questions asked in this scenario will be answered in the document updates.

Sneaky Job Update

As a variation of the previous scenario: Start an application by scheduling resources (e.g. batch queues), but schedule on more than one resource set and cancel remaining schedule times after one is activated (or reuse for another purpose?).

```
user ---> portal ---> resource broker
resource broker ---> | I | find range of resource sets
                     | N | schedule range of resource sets
                     | F | stage application prerequisites on range of
                           resource sets
                     | O | <application starts>
                     |   | cancel scheduling on any excess resource sets
```

Notes on Job Submission

Information and Job ID

The previous examples don't explicitly mention the relation between job submission and information services. Key requirements/questions here are:

- Each job needs a unique ID. QUESTION: Who issues this? It should be unique for the entire world!
- Each job should be related, by this unique ID, to any parent or child processes, to its associated data etc.
- Information services should be able to find the state of the job at anytime (scheduled, running, moving, estimated to run when, ...). Either the resource broker should be providing this information, or a subset of it to the information services, or should the information services be pulling it from the resource services?

2.2.2 Application Migration Scenario

```
user ---> portal ---> resource management service
                  ---> application
application ---> migrates itself
             ---> resource management

resource management service ---> | | migration strategy [*]
                                 |I| find new resource set [*]
                                 |F| schedule new resource set
                                 |N| instruct application to checkpoint
                                 |O| [and maybe shutdown]
```

```
                                  | | <application checkpoints>
                                  | | transfer appropriate checkpoint
                                      file/s
                                  |I| to resource set [*]
                                  |N| get application prerequisites on
                                  |F| new resource set
                                  |O| <application starts on new resource
                                  | | set>
                                  | | shut down old job if necessary
migrates itself  ---> application makes own explicit calls to various
service
```

Motivation

Many large scale simulation in the field of hydrodynamics or meteorological modeling have compute time requirements, which go way beyond the queue time limits of supercomputers, or which cannot even be predicted at the start of the simulation. This limitation requires the researcher to start the tedious process of securing the simulation's checkpoint files, archiving them if necessary. If the simulation is continued on another host, checkpoint files need to be transferred and the simulation is resubmitted to the queuing system. At all steps, the user's manual interaction makes the process prone to failure: Checkpoint files can be erased by disk quota timeouts, the manual transfer of data takes long and resource requirements have to be correctly analyzed. Last but not least, the researcher is required to remember usernames and passwords as well as interface with a wide range of different machines, architectures, queuing systems and shell programs.

The overhead of this procedure often lets to researcher resubmit on the same machine, where the simulation has been run before. The new simulation will be queued and precious time is wasted by waiting to be served.

The process of resubmitting to an arbitrary host, which fulfills minimum resource requirements is a prime candidate for automation. In nomadic migration scenario, the application profiles its performance and provides data such as disc usage or maximum memory requirements. Since the application is moderately aware by which time, the queue time is about to expire, it will engage in a checkpointing procedure and inform a Migration Service about the upcoming migration. The migration server receives information on the checkpoint files, executable and resource requirement. It will lookup an appropriate resource, stage checkpoint and executable to the new host and submit. The appropriate communication and queue system language is automatically chosen, previously written output data is transferred to a user given destination.

Notes on Application Migration

migration strategy

- Is migration possible? (e.g. enough disk space, possible to stream data, can application checkpoint?)

- Is a checkpoint file needed? (won't be needed for all applications)

find new resource set

- this should take into account the size of any checkpoint files which need to be moved to the resource set for restarting

transfer checkpoint file/s

- this has to include any recombination etc of data files, as dictated by Application Description System.

order of events

Depending on needs/instructions there are actually many possibilities:

```
schedule new resource set
Optional: make reservations on the new resources
instruct application to checkpoint
instruct application to migrate
kill old application when it start running on a new resource
```

requirements on applications

- Application must be checkpointable!
- Application independent of calculated data (maybe don't really need this, this dependency could just be added to job description)
- Application can produce checkpoint file on demand (may even be possible to do something even if not produced on demand, e.g. use last checkpoint file).

2.2.3 Scenario: Migration Decision

The decision to migrate could be made by a user, by the application itself, or by the resource management service. In general making a decision is not a trivial problem. In the grid environment, where typical user has usually lower priority than the local user, appearance of the latter one's jobs can caused a serious delay in job processing. This and other factors should be taken into consideration while making the decision about a migration. This problem, however, will be considered in another paper. In each case a decision generates a signal to the system to migrate. The following scenarios are made possible within GridLab project.

- Resource management service initiates migration.
  Why? the resource management service will contain the most complete information and decision making abilities for the simulation, and will be able to make decisions based on many different criteria. (Note though that depending on the flexibility of the resource management service, the portal may have additional

information about a given virtual organization). The resource management service may decide to migrate based on:

- Contract violation
- Information from monitoring system
- Information about resource set
- More appropriate (cheaper/faster/larger) resource set
- Management of group resources

How? The resource management system initiates the migration process detailed in Migration Scenario section.

- User/portal initiates migration
  Why? Demonstrations/testing, user has some information about the resource set (machine will be going down, disk space insufficient for output), portal has better idea of how to manage group resources.
  How? user initiates migration from a portal interface, or portal initiates migration on behalf of the user. The portal should signal the resource management service to migrate the simulation, or alternatively signal the simulation to migrate itself.
- Simulation initiates migration
  Why? simulation receives signal from operating system that machine is going down or queue time is nearly over, simulation realizes that it requires additional resources to continue (memory, CPU, disk space).
  How? the simulation can signal the resource management service to migrate the simulation, or alternatively migrate itself.

These and other scenarios will be made available by developing the GRMS system. In the next section we define the system components which will participate in the GridLab resource management. The architecture of the system will be updated continuously throughout the project lifetime.

3. Description of GRMS architecture and its components

This chapter outlines the requirements of each service of the GRMS system service domain. Figure 1 depicts the overall architecture of these services.

With respect to functionality, the GridLab application community, and thus, the GRMS service domain needs the following services:

• ***System configuration management***: For all instruments in the system, CIM based functional and performance monitoring is necessary. Monitoring should be done in a timely fashion and events should be delivered according to client set policy or criteria.

• ***Job execution management***: Time, priority, and space based scheduling of jobs. In case of application failure, jobs are retried based on applicable policy.

• ***Resource management:*** Dynamic and flexible resource management is essential. At the same time resource isolation between different jobs is crucial, not only for access control but also to ensure that there are no unexpected performance dependencies.

• ***Infrastructure services***: user management, accounting management (these services are out of scope of the GridLab project. We plan to get these services from other Grid projects, either European or US/Asia-Pacific. Close collaboration with CrossGrid project guarantees the deployment of these services within GridLab), logging and tracing.

3.1 GRMS Configuration and Policy Services

GRMS Configuration service is responsible for GRMS configuration. This service allows users (GRMS administrators) to enforce different resource management policies which can be then mapped to different groups of users. Moreover, the basic configuration of GRMS will be supported by this service, including selection of various scheduling algorithms and strategies.

Additionally, ***GRMS Policy Service*** has been introduced to handle a wide variety of policy needs, with a special focus on policies governing resource allocation in a particular Virtual Organization. The policy service will support single domain policy management as well as multi-domain policy management and control. In such a case the policy service will discover and take into account the policies of different (virtual) organizations as part of their resource publishing and matching process.

Both kinds of services will be developed in the last stage of the GridLab project.

3.2 Core GRMS Services

Pure GRMS services include Job Receiver, Resource Discovery, Resource Evaluation, Brokering, Prediction, QoS, Resource Reservation and Resource Estimation services.

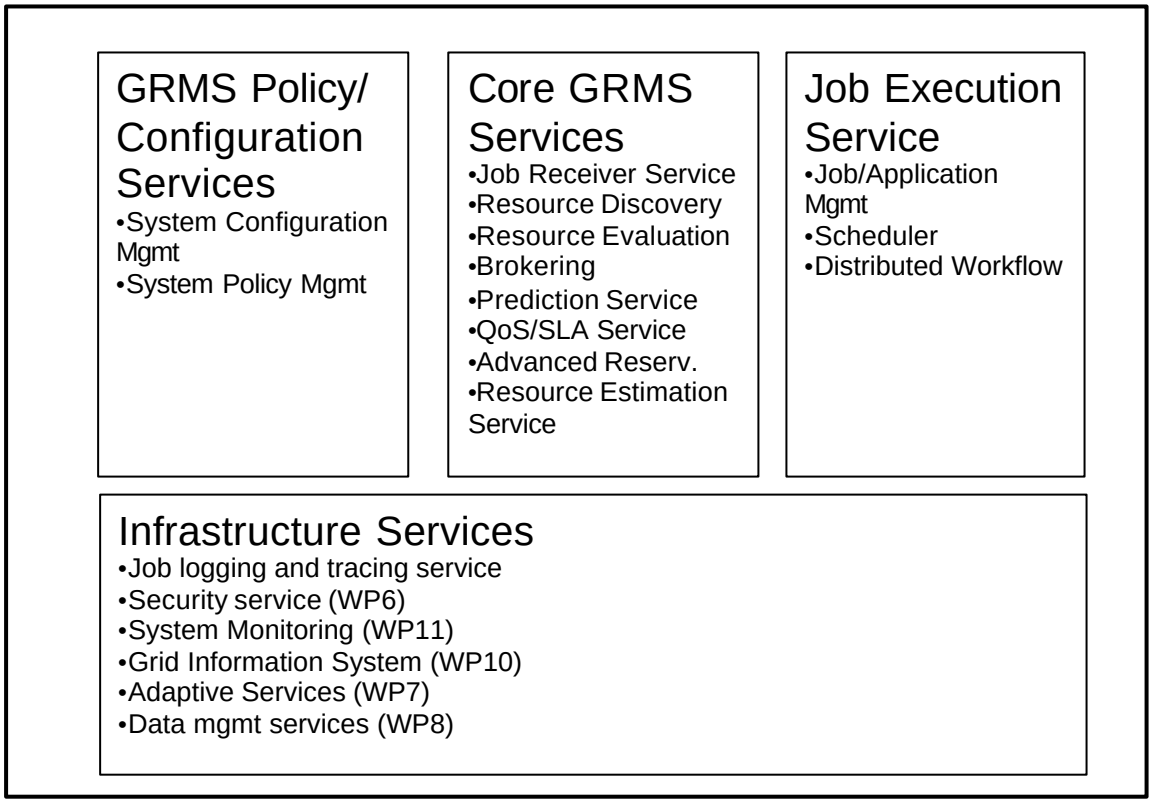


Figure 1: GRMS fundamental services against other GridLab services

Job Receiver (JR) is responsible for receiving the job requests and queuing them for the time of processing the first job request sitting in a queue. The Job Receiver Queue (JRQ) is a distributed service (there might be many instances of this service) with a notion of job arrival time. It means that there are no job requests coming to two different Job Receiver services at the same time.

Resource Discovery Service (RDS) is a service which discovers potentially good resources that could be allocated for the given job request. In general, RDS returns unranked list of the resources for a particular job request. Those resources will be further evaluated by the ***Resource Evaluation Service.***

Resource Evaluation Service (RES) is responsible for finding out which of the given resources are the best ones from the point of view of the user's job request. RES takes into account such parameters as:

- Quantifiable User Criteria (cost and time criteria)
- Non-quantifiable User Criteria (resource architecture, operating system etc.)

- Preferences among the user criteria
- Application performance characteristics
- Historical data about completed jobs (history based predictions)
- Statistical information
- Resource characteristics
- and many more.

RES is supported by intelligent decision making system and can also be supported by the ***Adaptive Component Service (ACS)*** provided by WP7 of the GridLab project. RES will be elaborated more in future GridLab conference papers. We will strictly distinguish between production and research based components of the RES in the future work.

Brokering Service (BS) is a central service of the GRMS. It allows for allocation of the resources to the user jobs. Once the job is submitted a broker, together with a ***Job Manager Service (JMS)*** and ***Distributed Workflow Service (DWS)*** controls the job. Broker can send various signals to a job, such as Job_Checkpoint, Job_Kill, Job_Migrate, Job_Stop etc. The signals can be sent automatically by the Broker or can be enforced by the Job Manager or directly by a user. Such a behavior allows implementing many dynamic application scenarios required by GridLab applications (see section 2 for more details).

Prediction Service (PS) is responsible for short time predictions of the resource behavior as well for job run rime and queue wait time predictions. The prediction service will reason about the jobs based on the previous jobs (or similar jobs) runs. The levels of using prediction mechanisms in the GRMS will change according to the future prove of this technology. During the first project stages the ***Logging and Tracking Service (LTS)*** will gather information about the jobs and store this information in the local (to GRMS) data base. This information will be used in the future for history based predictions. We will strictly distinguish between production and research based components of the LTS in the future work.

Advance Reservation Service (ARS) is responsible for making advance reservations of resources to make them available for jobs in a given, but negotiated with the service, time frame in the future. This functionality is possible on only several local resource management systems. This service is to be implemented by NTUA from Greece.

QoS Service (QOS) is a service which allows a user to negotiate the quality of service provisioned to the job. This functionality is considered as a pure research activity within GridLab.

Resource Estimation Service (REST) may be used to predict, for a particular job, what kind of resources are needed to process a job in an absence of the proper job description.

So, the REST will produce a resource description (file). This will be an ASCII text file using a meta-language that can be interpreted by the BS and the scheduler.

3.3 Job Execution Service

Job Execution Service addresses resource scheduling and computing access. The ***Job Management Service (JMS)***[1] must deal with the three types of job requests (Java Programs, Batch Process and System Synthesis). The JMS must manage the jobs during their lifetime. It carries out initial processing needed to prepare jobs for execution, submits jobs to the scheduler, and carries out any necessary post-processing.
The ***Scheduler*** must arrange for job execution based on job requirements, e.g., time, priority, cost, place, user preferences etc. The scheduler must maintain jobs and their schedule in a persistent scheduling database. The ***Distributed Workflow Service (DWS)*** must manage job workflow based on standard workflow specifications (e.g. WSFL, XLANG, and WSCI).

With respect to functionality, the GridLab application community, and thus, the GRMS service domain needs the following services:

- ***System configuration management***: For all instruments in the system, CIM based functional and performance monitoring is necessary. Monitoring should be done in a timely fashion and events should be delivered according to client set policy or criteria.
- ***Job execution management***: Time, priority, and space based scheduling of jobs. In case of application failure, jobs are retried based on applicable policy.
- ***Resource management:*** Dynamic and flexible resource management is essential. At the same time resource isolation between different jobs is crucial, not only for access control but also to ensure that there are no unexpected performance dependencies.
- ***Infrastructure services***: user management, accounting management (these services are out of scope of the GridLab project. We plan to get these services from other Grid projects, either European or US/Asia-Pacific), logging and tracing.

4. Architectural Representation

This section describes what software architecture is for the current system, and how it is represented. Of the **Use-Case**, **Logical**, **Process**, **Deployment**, and **Implementation Views**, it enumerates the views that are necessary, and for each view, explains what types of model elements it contains.

[1] The Triana-Grid Group (WP3) has taken the leading role in the software development of the *Application Manager* for GridLab. The *Application Manager* was not defined in the original proposal but it has become increasingly apparent to several partners that such a manager was necessary. This work will be done in close cooperation between all WPs, including WP9.

4.1 Architectural Goals and Constraints

To describe the architecture, there must be some assumptions defined, which specify characteristics of whole model, and each layer.

- General rules concerning layers:
  - There are three layers of the GRMS architecture: User Access Layer, Services Layer and System Layer
  - Each component of the GRMS architecture belongs to one layer
  - in each layer there are one or more components
  - components from one layer can communicate only with components from its own and neighboring layer (e.g. module from User Access Layer communicates only with modules from Services Layer)

- Functionality of individual layers:

1) User Access Layer (application layer) ;
   - in this layer user applications are placed;
   - applications are developed using appropriate API which gives them access to Services Layer;
   - application can not communicate with GRMS services layer directly;

2) GRMS Services Layer
   - the layer where Grid Environment Resource Management "intelligence" is placed;
   - layer is comprised of modules called services which are network-enabled entities that provide some capability;
   - functionality of services is forced by requirements of applications;
   - services exploit System Layer to provide their functionality;
   - services provide well-defined, consistent interface through which are accessed;
   - services can be accessed by application (or portal) or by other service;
   - scalability requirement: it is possible to add new service without any changes in rest of services, and to add new instance of the same service (fault tolerance requirements);
   - Services Layer should provide mechanisms of service discovery – application do not have to be aware of all available services at development stage but can discover services during runtime
   - should provide strong fault tolerance mechanisms;
   - should be implemented using technology that supports scalability and interoperability
   - security is enforced by the WP6 system architecture;

3) System Layer
   - provides basic grid infrastructure with which GRMS Services Layer interacts;

- component placed in this layer can be accessed only by components from Services Layer (in general can not be accessed directly from User Access Layer; other situations are possible but they imply non-secure resource management);
- generally most of System Layer components can be recognized as elements which have to be installed on individual machine (but not necessary);

- Consequences
  - choosing some architecture means to apply all rules (assumptions) it brings in implementation specification;
  - it is much better to remove some assumptions now than breaking them during system design or implementation stage;
  - from implementation point of view access to middleware layer is realized in one technology (e.g. web services, CORBA, OGSA);

4.2 Use-Case View

A use-case model is a model of the system's intended functions and its surroundings, and serves as a contract between the customer and the developers. Use cases serve as a unifying thread throughout system development. The most important purpose of a use-case model is to communicate the system's behaviour to the customer or end user.
The use-case model allows software developers and end users to agree on the requirements. It is a model of a system containing actors, use cases and their relationship.
Use-case diagrams generally presents user perspective view on system (user level use-case diagram), but it is also possible to model system behaviour using that diagrams (developer level use-case diagram) .

User Level Use-Cases.

User level use-case presents user view on functionality provided by GridLab Resource Management System.

Actors

- **Application**
  Application is a general term describing entity which utilizes functionality provided by GridLab Resource Management System. It is the only actor that initiates all use cases for GRMS. From the implementation point of view Application is an implementation of GAT API. As the Application also portal is considered.

- **Authorization System**
  Authorization System is a system which provides security mechanisms in user access to GridLab Resource Management System. This actor represents system

which is used by GRMS.

- **File Transfer System**
  File Transfer System is a system provided by WP8 which is responsible for transferring data between sites in GridLab project. This actor represents system which is used by GRMS.

- **Information System**
  Information System is a system that provides all needed (static and dynamic) information about resources. This actor represents system which is used by GRMS.

Use-Cases

- **Check Job Status**
  Use case begins when Application calls a specific function of checking job status. Operation should be authorized first. Then the systems checks job's status according to job id provided by Application.

- **Estimate Resources**
  Use case begins when Application calls a specific function of estimating resources. The system estimates the resource needs for job (e.g. what resources are needed to finish job before deadline)

- **Find Best Resource**
  Use case begins when Application calls a specific function of finding best resource. At first this operation should be authorized in Authorization System. Then the GRMS evaluates resources which are provided by Information System. The system returns chosen resource and use case ends.

- **Migrate Job**
  Use case begins when Application calls a specific function of job migration. At first operation should be authorized in Authorization System. The system (Gridlab Resource Management System) takes - provided by Application - job migration description and according it submits job. The File Transfer System is used to copy files to new location.

- **Predict Job Execution**
  Use case begins when Application calls a specific function of predicting job execution. According to job description given by Application the system predicts job execution (start time, end time, run time) on specific resource

- **Remove Job**

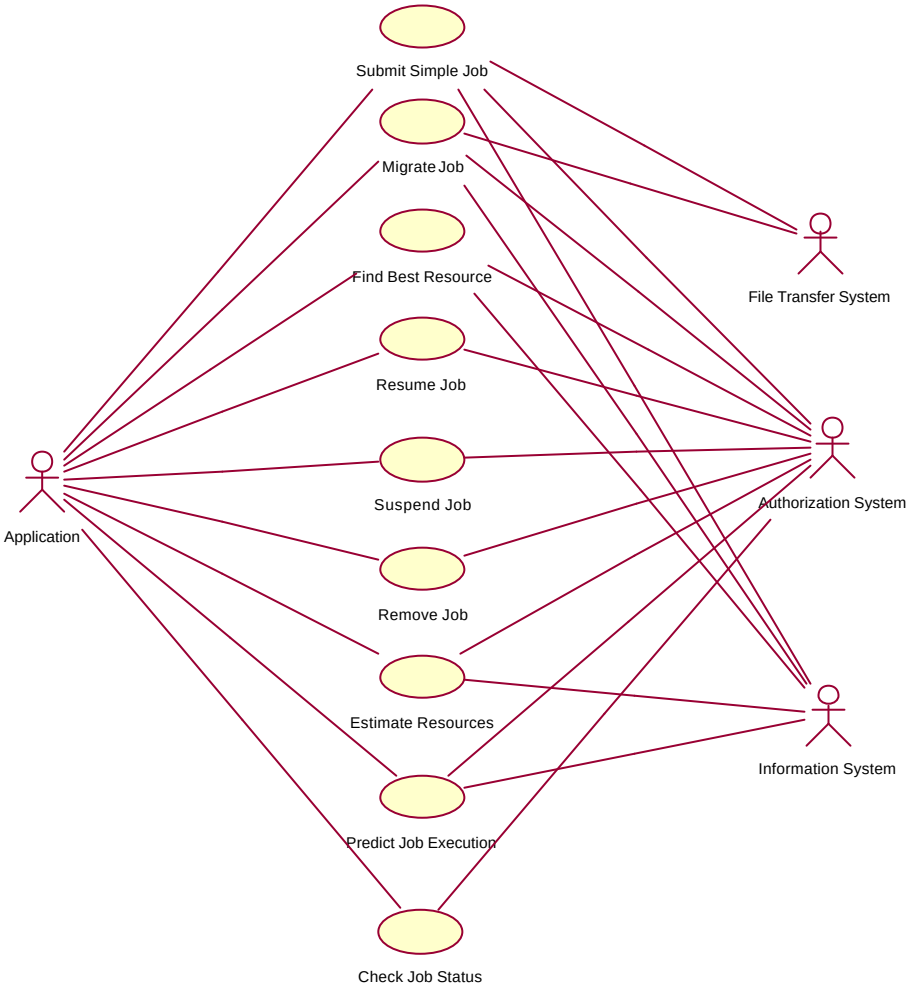
Use case begins when Application calls a specific function of removing job. Operation should be authorized first. Then system checks relationships with other jobs, and if it is allowed the systems removes job of id provided by Application.

- **Resume Job**
  Use case begins when Application calls a specific function of resuming job. Operation should be authorized first. Then the systems resumes job of id provided by Application.

- **Submit Simple Job**
  Use case begins when Application calls a specific function of job submission. At first operation should be authorized in Authorization System. The system (Gridlab Resource Management System) takes - provided by Application - job description and according it submits job. For finding resources Information System is used. Depending on job description the system can interact with File Transfer System for coping files. The system returns job identifier to Application and use case ends.

- **Suspend Job**
  Use case begins when Application calls a specific function of suspending job. Operation should be authorized first. Then the systems suspends job of id provided by Application.

User Level Use-Case Diagram



4.3 Developer Level Use-Cases

Developer Level Use Case diagrams presents use cases from system perspective.

Actors

- **Application**

Application is a general term describing entity which utilizes functionality provided by GridLab Resource Management System. It is the only actor that initiates all use cases for GRMS. From the implementation point of view Application is an implementation of GAT API. As the Application also portal is considered.

- **Authorization System**
  Authorization System is a system which provides security mechanisms in user access to GridLab Resource Management System. This actor represents system which is used by GRMS.

- **File Transfer System**
  File Transfer System is a system provided by WP8 which is responsible for transferring data between sites in GridLab project. This actor represents system which is used by GRMS.

- **Information System**
  Information System is a system that provides all needed (static and dynamic) information about resources. This actor represents system which is used by GRMS.

Use-Cases

- **Analyze Job Description**
  This use case occurs when system needs to analyze job description provided by Application

- **Analyze Prediction Info**
  This use case occurs when system uses prediction mechanisms for resource evaluation.

- **Authorize**
  This use case occurs at the beginning of every operation with is performed by the Application on GRMS. To authorize user Authorization System is used.

- **Check Job Status**
  This use case occurs when Application wants to check status of previously submitted job. Related use cases: Authorize, Get Id On Resource

- **Estimate Resources**
  This use case occurs when Application wants to estimate resources which are needed for job execution. Related use cases: Authorize, Find Potential Resources, Analyze Prediction Info

- **Evaluate Resources**
  This use case occurs when system wants to choose the "best" resource among potential ones which match description provided by the user

- **Find Best Resource**
  This use case occurs when Application wants to get the "best" resource which matches provided description.

- **Find Executable**
  This use case occurs when system tries to find job's executable based on name provided by the Application.

- **Find Potential Resources**
  This use case occurs when system wants to get list of resources that matches description provided by the Application

- **Get Job From Queue**
  This use case occurs when system gets job from Job Queue for execution.

- **Get Job Id On Resource**
  This use case occurs when job id on local resource management system is needed.

- **Migrate Job**
  This use case occurs when Application wants to migrate a job. Migration is realized according do description provided by Application. Related use cases: Authorize, Analyze Job Description, Put Job Into Queue, Schedule And Run Migrated Job

- **Predict Job Execution**
  This use case occurs when Application wants to predict job execution. According to job description given by Application and using historical knowledge the system predicts job execution (start time, end time, run time) on specific resource.
  Related use cases: Authorize, Analyze Job Description, Find Potential Resources, Analyze Prediction Info

- **Put Job Into Queue**
  This use case occurs when system puts job into job queue.

- **Register Job Id**
  This use case occurs when system wants to make change in Job Id stored in Job Id Repository.

- **Remove Job**
  This use case occurs when Application wants to remove previously submitted job. Related use cases: Authorize, Get Job Id On Resource

- **Resume Job**
  This use case occurs when Application wants to resume previously suspended job. Related use cases: Authorize, Get Job Id On Resource,

- **Schedule And Run Job**
  This use case occurs asynchronously and periodically, and is needed for realization of "Submit Job" use case. Related use cases: Get Job From Queue, Find Potential Resources, Evaluate Resources, Transfer Files, Find Executable, Register Job Id

- **Schedule And Run Migrated Job**
  This use case occurs asynchronously and periodically, and is needed for realization of "Migrate Job" use case. Related use cases: Get Job From Queue, Find Potential Resources, Evaluate Resources, transfer Files, Find Executable, Register Job Id

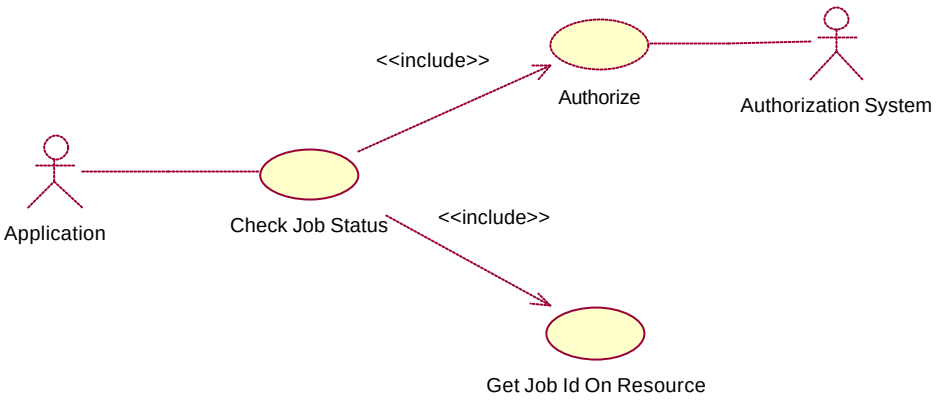
- **Submit Simple Job**
  This use case occurs when Application wants to submit a job. Submission is realized according do description provided by Application. Related use cases: Authorize, Analyze Job Description, Put Job Into Queue, Schedule And Run Job

- **Suspend Job**
  This use case occurs when Application wants to suspend previously submitted job. Related use cases: Authorize, Get Job Id On Resource

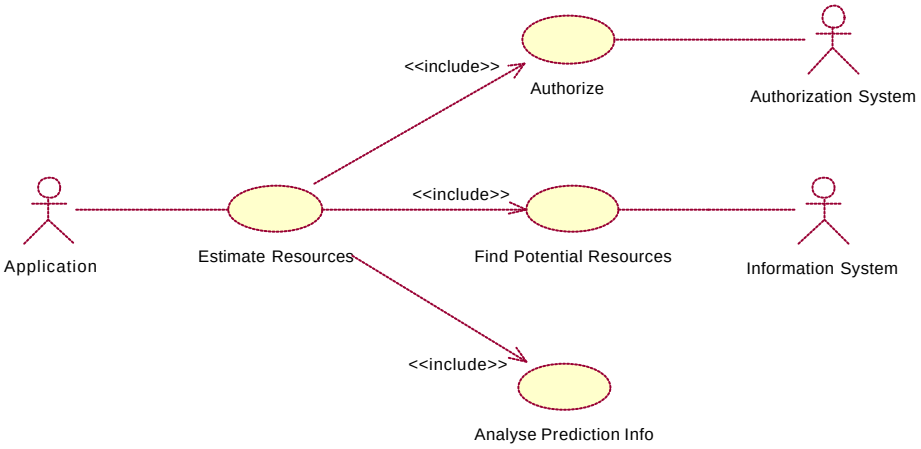
- **Transfer Files**
  This use case occurs when the system wants to move files from one location to another. To realize that task it contacts File Transfer System. Files which can be transfer are: data files, job's executables, migration data (job status).

Developer Level Use-Case Diagrams

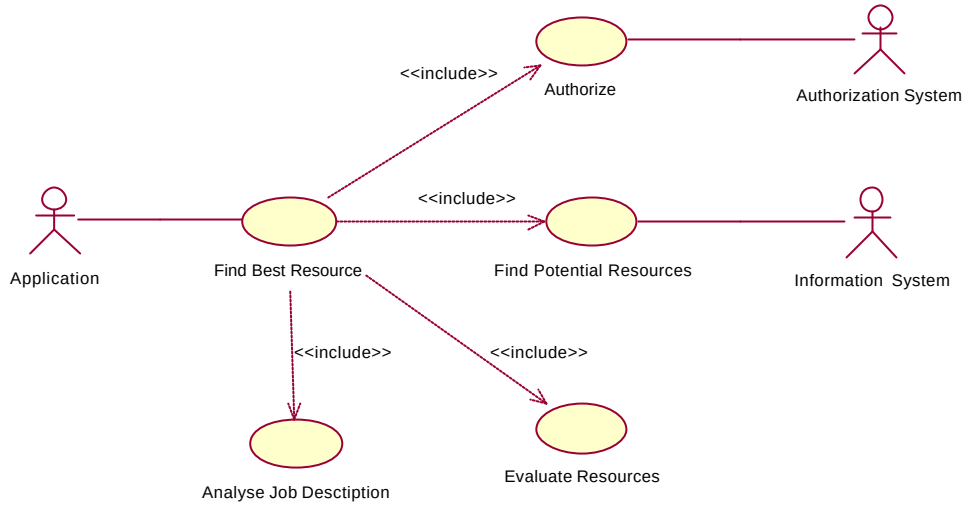
- Check Job Status Diagram



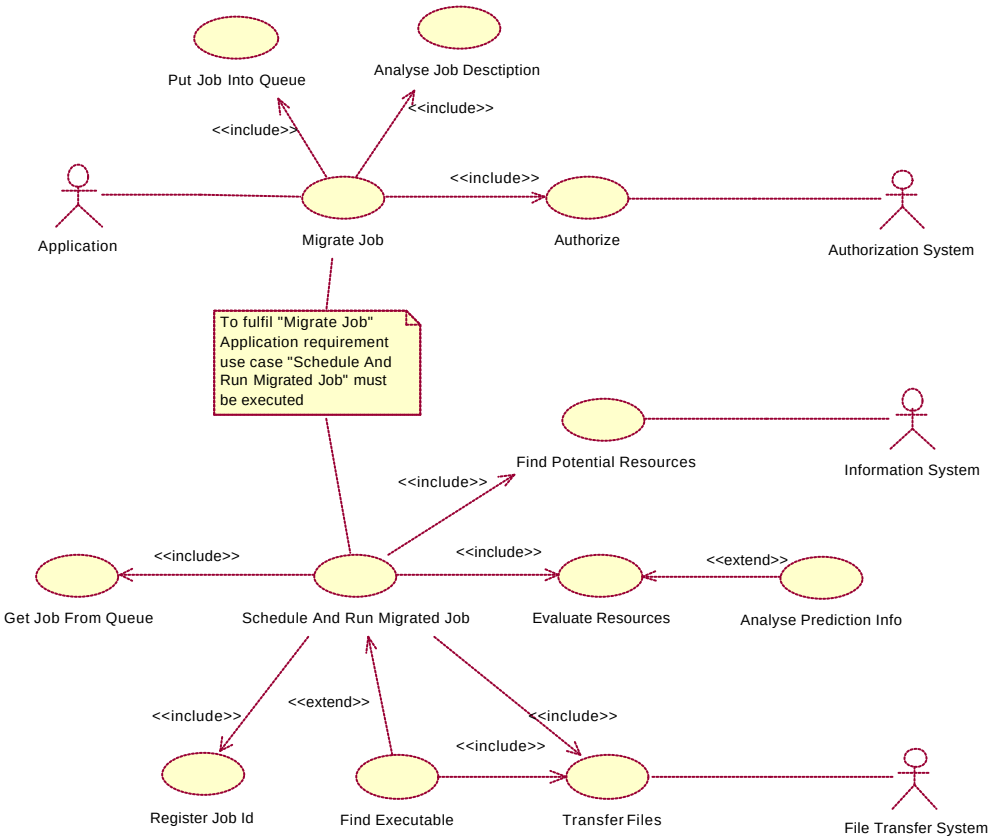
- Estimate Resources Diagram



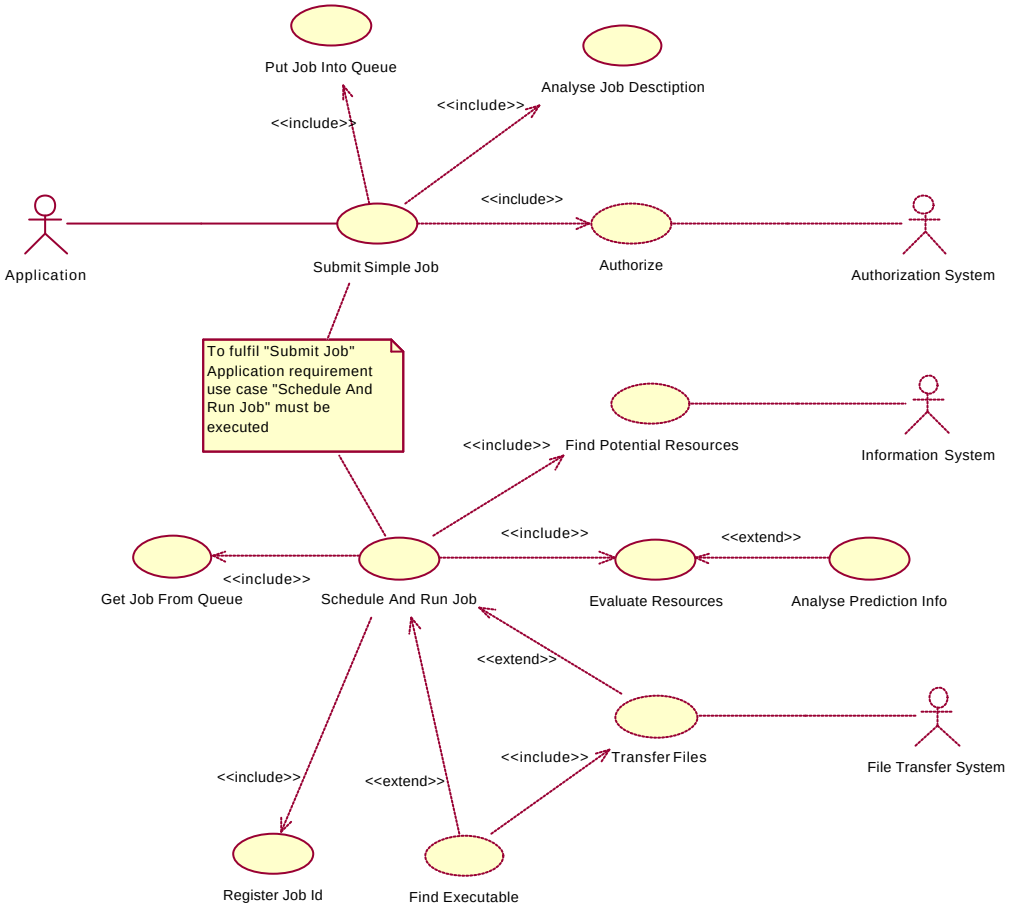
- Find Best Resources Diagram



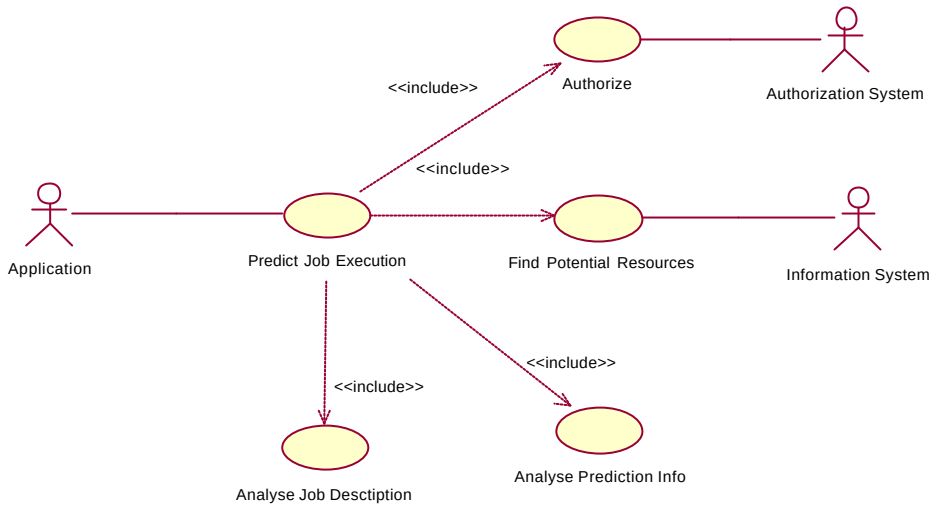
- Job Migration Diagram



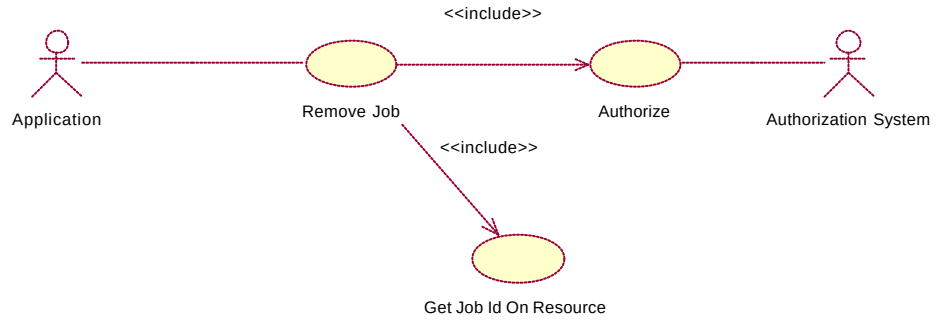
- Job Submission Diagram



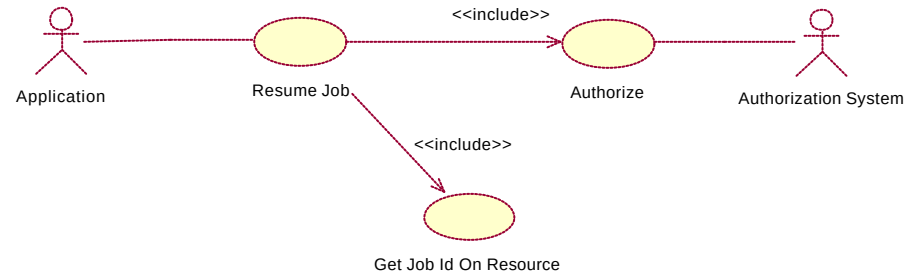
- Predict Job Execution Diagram



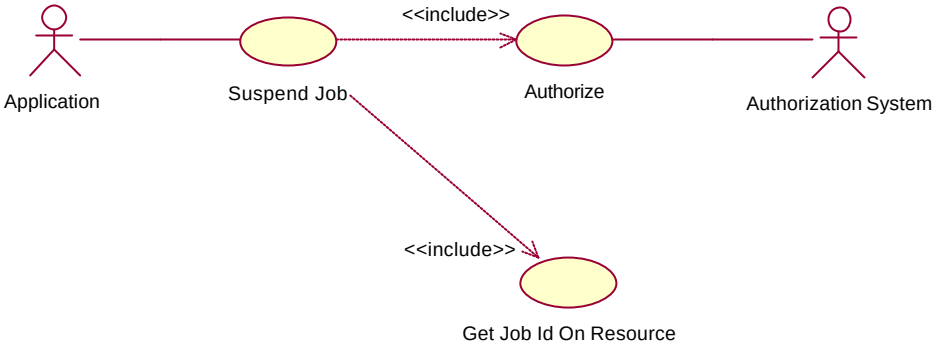
- Remove Job Diagram



- Resume Job Diagram



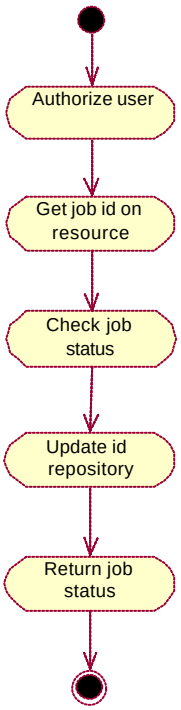
- Suspend Job Diagram



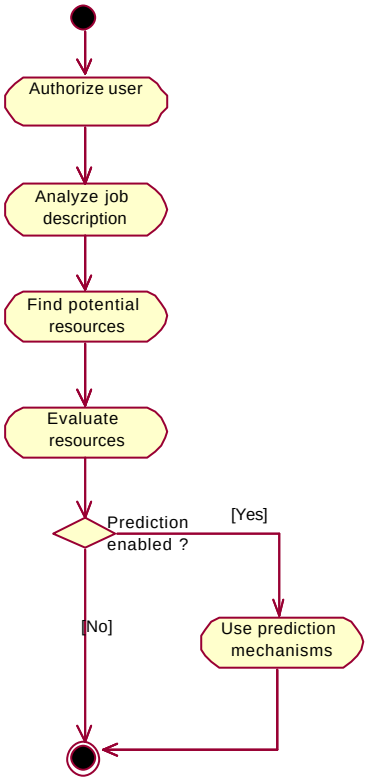
4.4 Activity Diagrams.

Activity diagrams presented here shows dynamics of the system. They represent the flow within particular use case from user-level use cases.

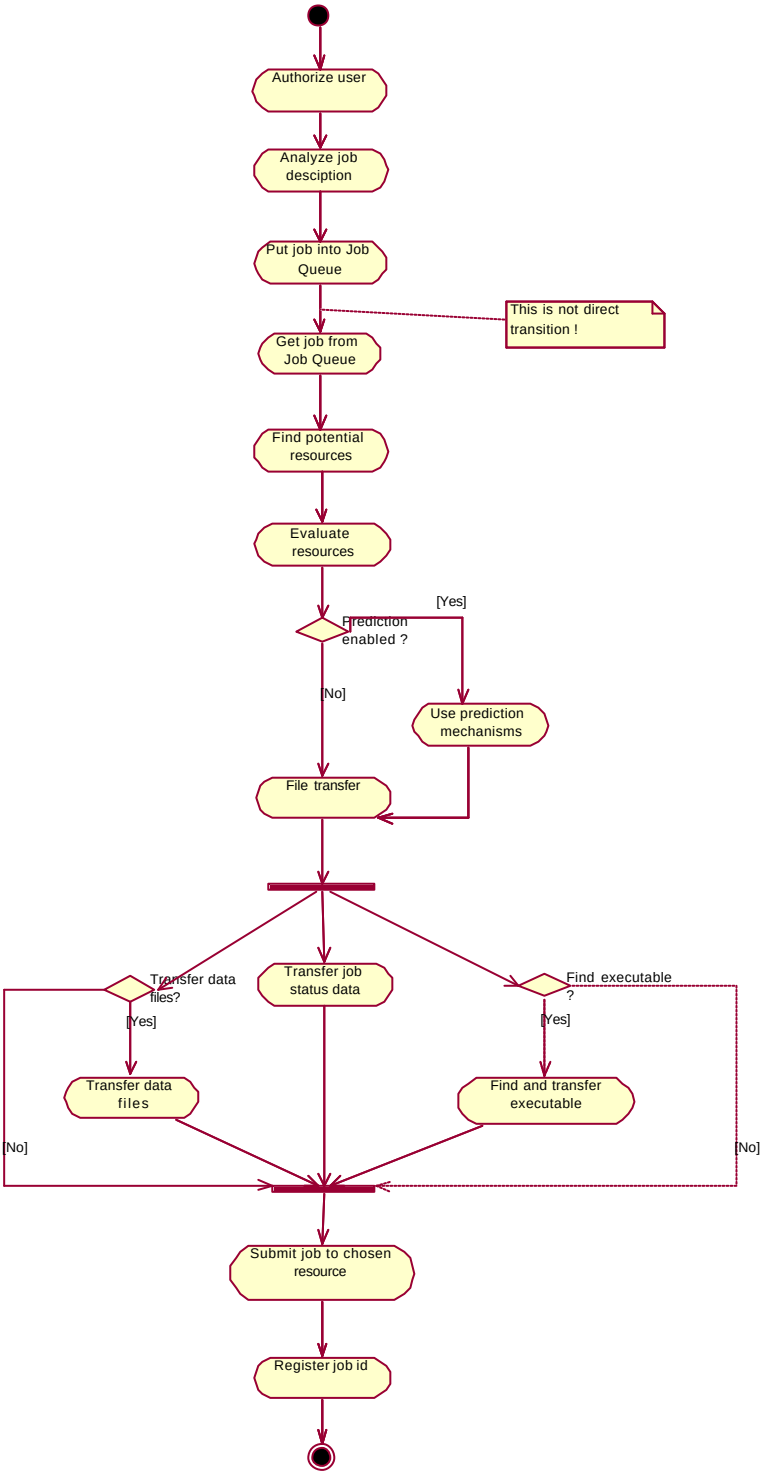
- Check Job Status Activity Diagram



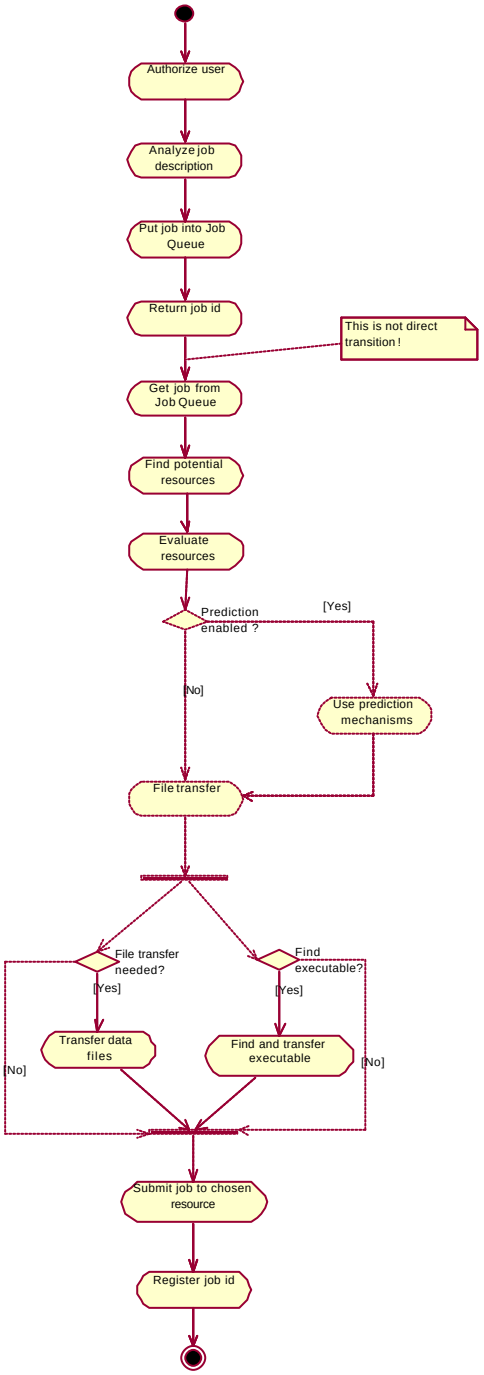
- Estimate Resources Activity Diagram
- Find Best Resources Activity Diagram



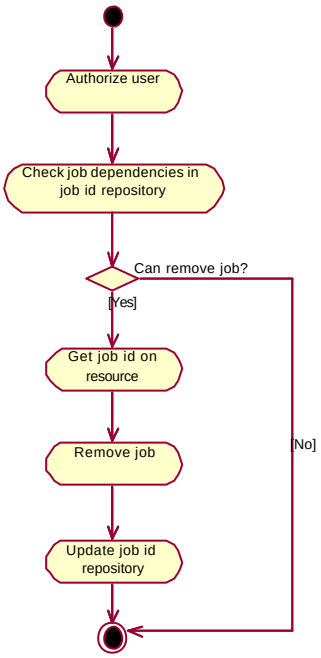
- Job Migration Activity Diagram


Authorize user
Analyze job desciption
Put job into Job Queue
This is not direct transition !
Get job from Job Queue
Find potential resources
Evaluate resources
[Yes]
Prediction enabled ?
[No]
Use prediction mechanisms
File transfer
Transfer data files?
Transfer job status data
Find executable ?
[Yes]
[Yes]
Transfer data files
Find and transfer executable
[No]
[No]
Submit job to chosen resource
Register job id

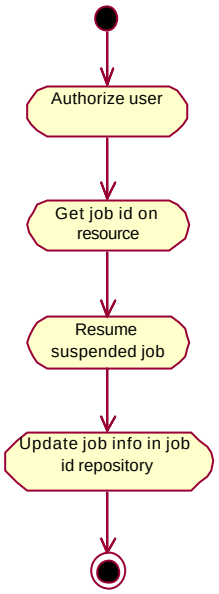
- Job Submission Activity Diagram


Authorize user
Analyze job description
Put job into Job Queue
Return job id
This is not direct transition!
Get job from Job Queue
Find potential resources
Evaluate resources
Prediction enabled ?
[Yes]
[No]
Use prediction mechanisms
File transfer
File transfer needed?
Find executable?
[Yes]
[Yes]
Transfer data files
Find and transfer executable
[No]
[No]
Submit job to chosen resource
Register job id

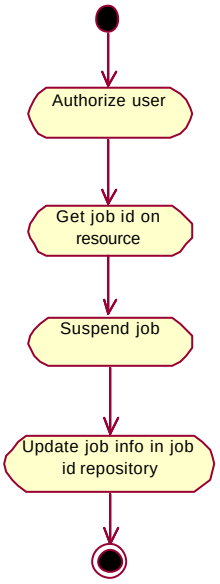
- Predict Job Activity Execution Diagram
- Remove Job Activity Diagram



- Resume Job Activity Diagram



- Suspend Job Activity Diagram

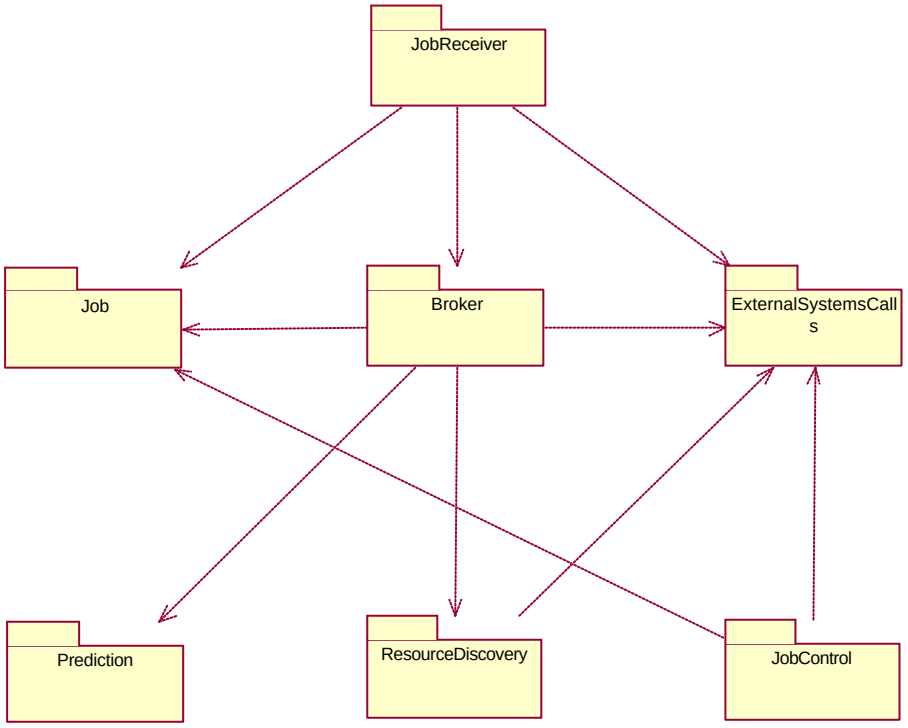


4.5 Logical View

This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. And for each significant package, its decomposition into classes and class utilities. architecturally significant classes are introduced and their responsibilities are described, as well as a few very important relationships, operations, and attributes.

Overview

Diagram below presents packages and their relationship in GridLab Resource Management System



Design Packages

Broker

- **GettingJobFromQueue**
  GettingJobFromQueue is a set of activities connected with analyzing jobs waiting in JobQueue and choosing one for execution.

- **JobMigration**

JobScheduling is a class responsible for steering of a process of job migration. It asynchronously and in periodic way gets job for migration from queue and steers all actions needed to prepare job for execution on new resource, and submits that job.

- **ResourceEvaluation**
  Resource Evaluation is a class responsible for choosing "the best" resource from among set of resources matching description provided by Application.

- **ExecutableFinding**
  ExecutableFinding is a class responsible for finding appropriate executable for the application name provided by Application in job description.

- **JobScheduling**
  JobScheduling is a class responsible for steering of a process of job scheduling. It asynchronously and in periodic way gets job from queue and steers all actions needed to prepare job for execution on chosen resource, and submits that job.

ExternalSystemsCalls

- **PotentialResourcesCall**
  PotentialResourceCall is a class responsible for contacting Information Service in order to get list of resources which match resource specification defined in job description received from Application.

- **AuthorizationCall**
  AuthorizationCall represents calls to Authorization Service in order to perform user authorization.

- **FileTransferCall**
  FileTransferCall is a class responsible for communication with File Transfer System to move files from one location to another. There are two cases of using file transfer:
  - for transferring data files (e.g. standard input of job) from location given in job description to location where job is going to be executed
  - for transferring executable from location given in job description to location chosen in scheduling process.

JobControl

- **JobControlInterface**
  JobControlInterface is an interface class used by Application for checking job status, removing, suspending and resuming a job. The most important argument for all of that operations is job identifier which is one given to Application during job submission

- **JobRemoval**
  JobRemoval is a class responsible for removing previously submitted job.

- **JobResuming**
  JobResuming is a class responsible for resuming previously suspended job.

- **JobIdUpdate**
  JobIdUpdate is a class responsible for changing job status which is part of Job Identifier stored in Job Repository (job status is changed to "removed" after job removal, "suspend" after job suspending, "running" after job resuming).

- **JobSuspension**
  JobSuspension is a class responsible for suspension previously submitted job.

- **JobStatusChecking**
  JobStatusChecking is a class responsible for controlling actions connected with getting status of previously submitted job.

ResourceDiscovery

- **BestResourceFinding**
  BestResourceFinding is a class responsible for controlling activities connected with finding best resource:
  - user authorization
  - analyzing description of resource
  - getting potential resources which match provided description
  - evaluating resources

- **ResourceDiscoveryInterface**
  ResourceDiscoveryInterface class represents interface for finding best resource.

Prediction

- **PredictionAnalysis**
  PredictionAnalysis is a process used during resource evaluation (ResourceEvaluation class) which helps to chose "the best" resource using historical knowledge about previous executions of given job.

JobReceiver

- **JobMigrationInterface**
  Interface for job migration. The most important arguments of job migration provided by Application are: job description (which describes migration) and user credentials, GRMS job identifier.

- **JobSubmitionInterface**
  Interface for job submission. The most important arguments of job submission

provided by Application are job description and user credentials.

- **MigrationReceiving**
  Class which controls all actions performed during migrating job receiving:
  - calling authorization service
  - job description analyzing
  - putting job into Job Queue.

- **SubmissionReceiving**
  Class which controls all actions performed during receiving job for submission:
  - calling authorization service
  - job description analyzing
  - creating new job id
  - putting job into Job Queue

Job

- **JobDescription**
  JobDescription class is an internal representation of job described originally in a job description provided by the Application

- **JobQueue**
  JobQueue represents place where received by GRMS jobs are stored, and wait for scheduling

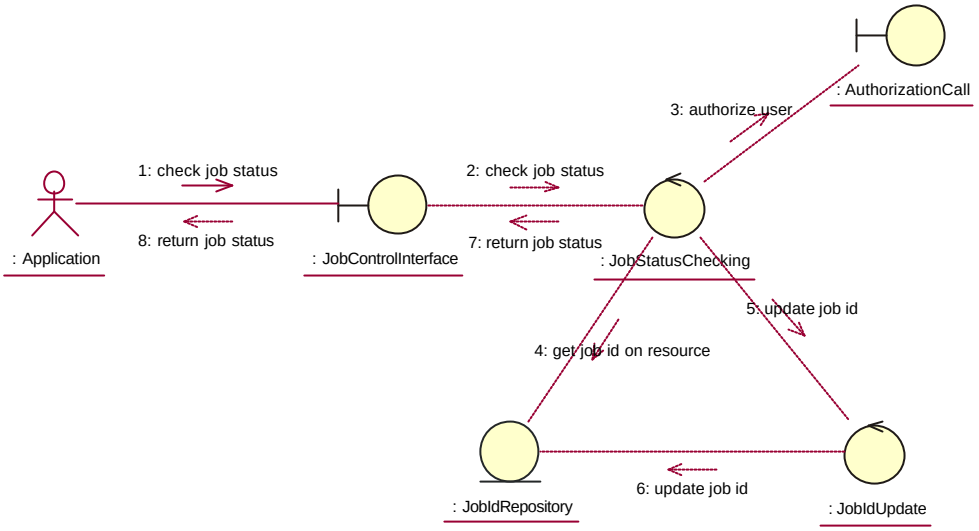
- **JobIdRepository**
  JobRepository is a place where JobIds are stored. It provides means to access individual JobId.

- **JobDescriptionAnalysis**
  JobDescriptionAnalusis is responsible for parsing job description provided by the Application, checking its correctness and creating internal representation of job (JobDescription class).

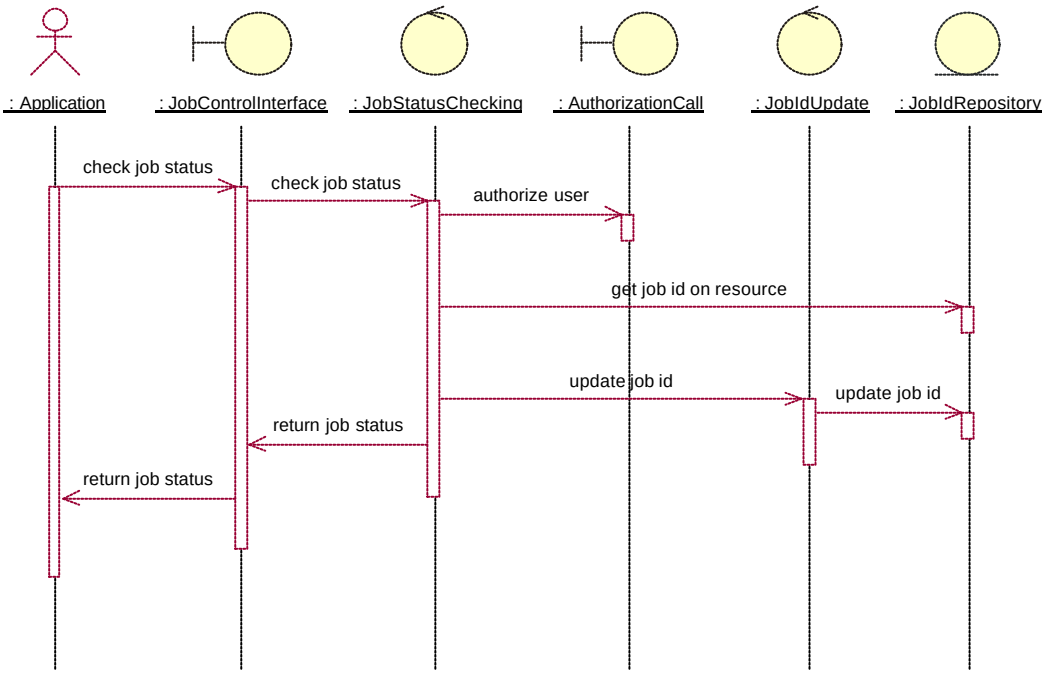
- **JobId**
  JobId is a class which represents job in GRMS. It is comprised of some elements:
  - unique GRMS job identifier: generated during job receiving
  - job identifier on resource assigned after submission job on resource
  - job status ("submitted", "migrated", "active", "suspended", "removed")
  JobIds are stored in JobIdRepository.

Use Case Realizations

- **Check Job Status**
  -Collaboration Diagram



- Sequence Diagram

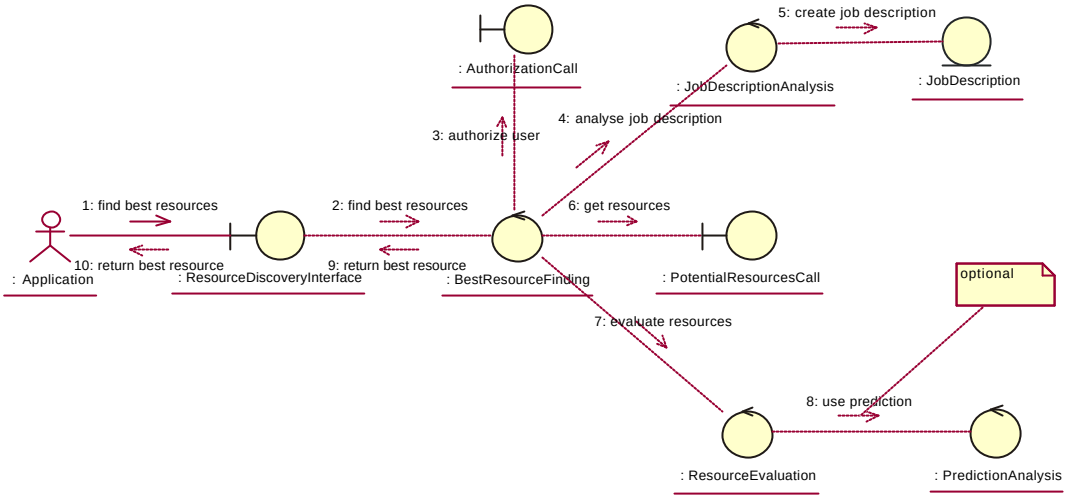


- **Estimate Resources**
  -Collaboration Diagram

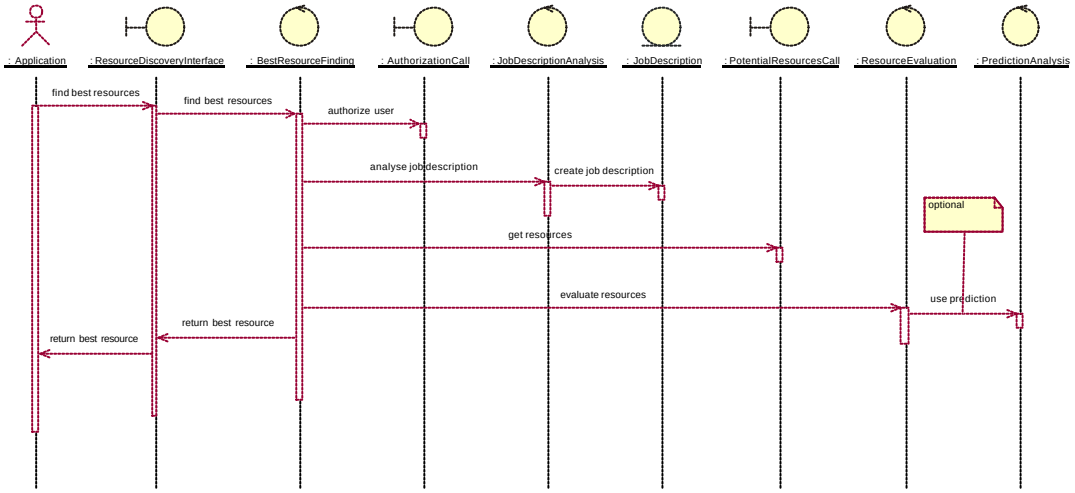
  - Sequence Diagram

- **Find Best Resources**

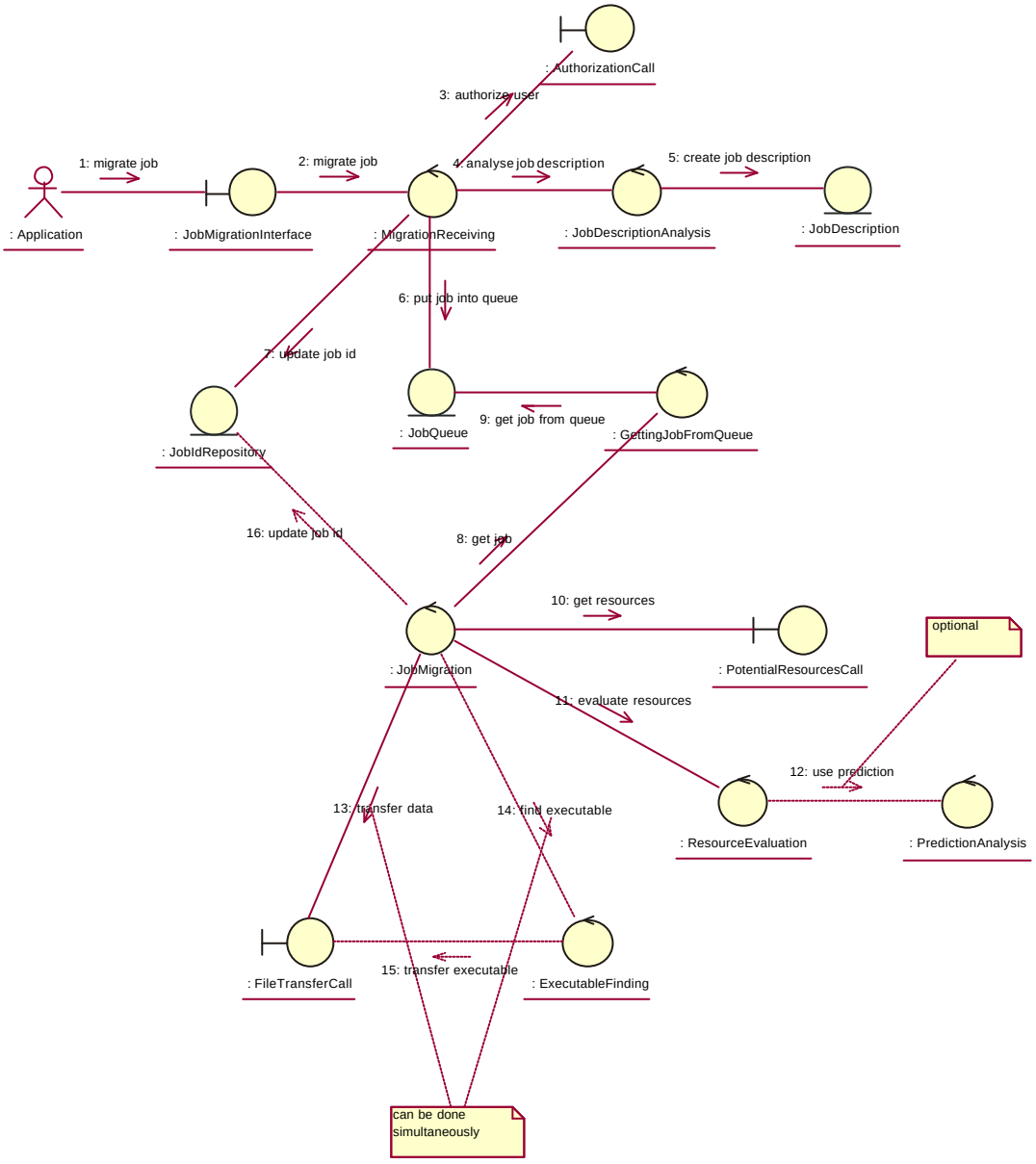
-Collaboration Diagram



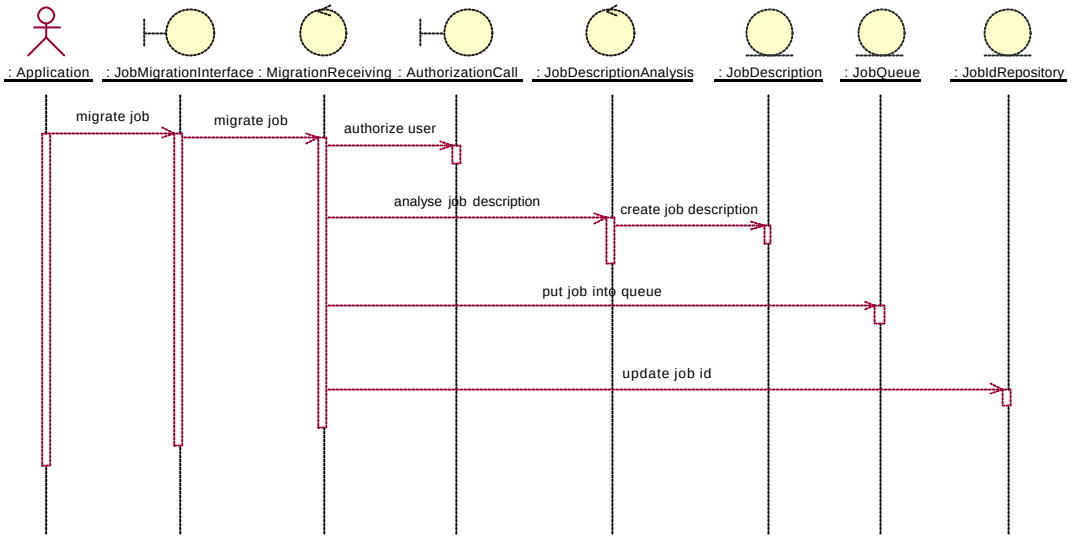
- Sequence Diagram

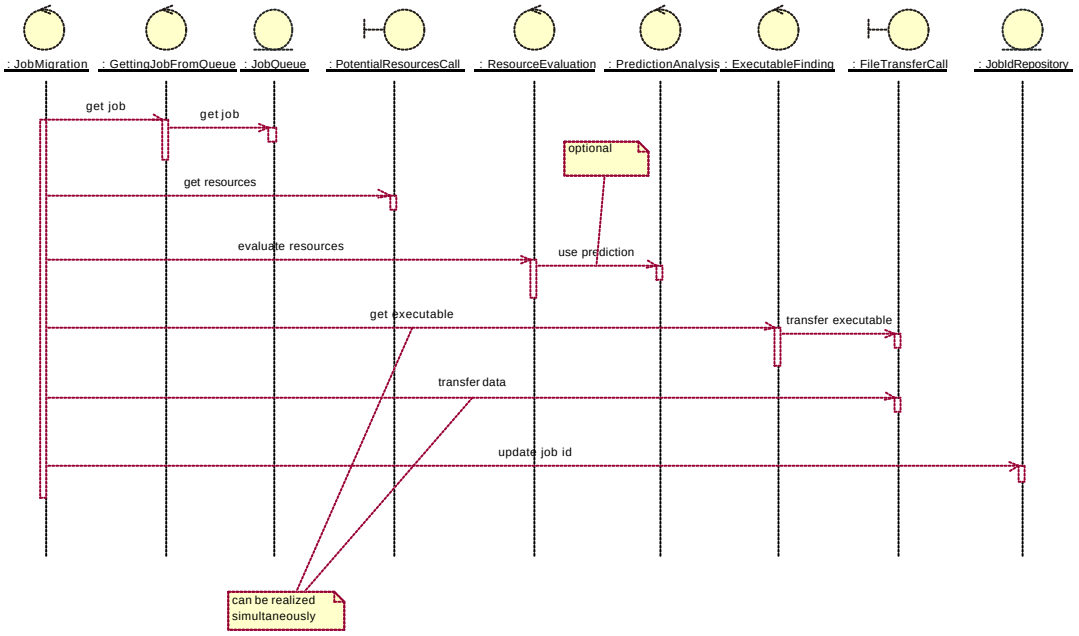


- **Migrate Job**
  -Collaboration Diagram



- Sequence Diagram

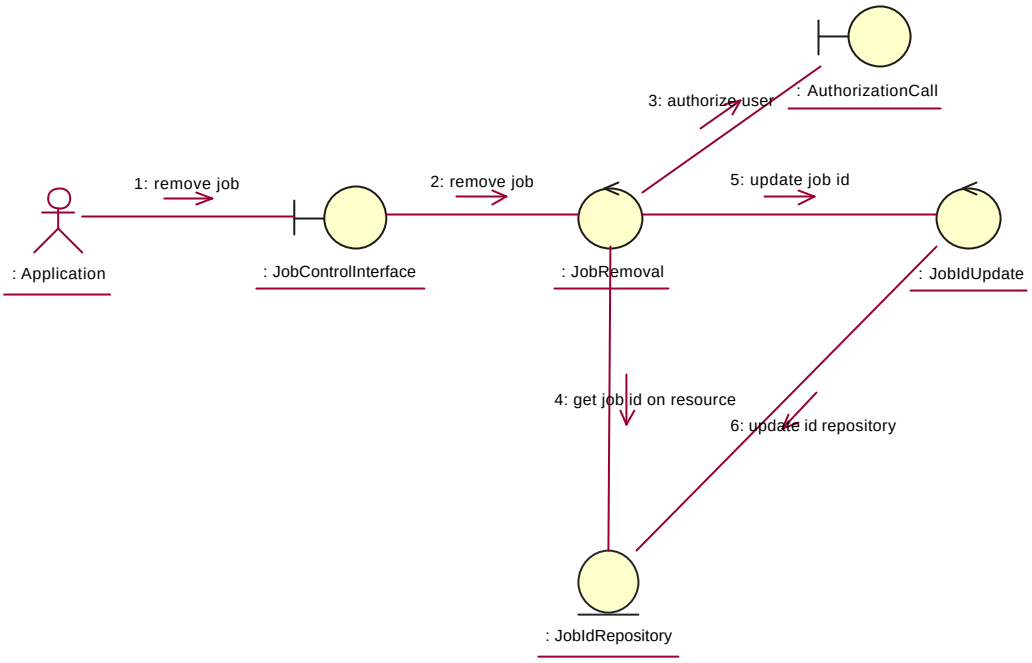




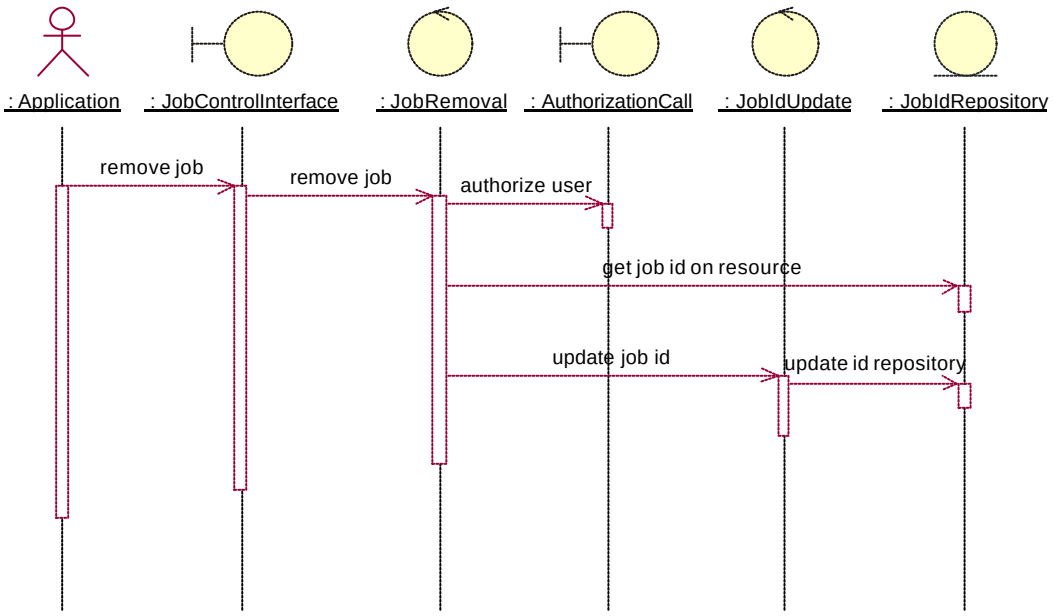
- **Predict Job Execution**
  -Collaboration Diagram

  - Sequence Diagram

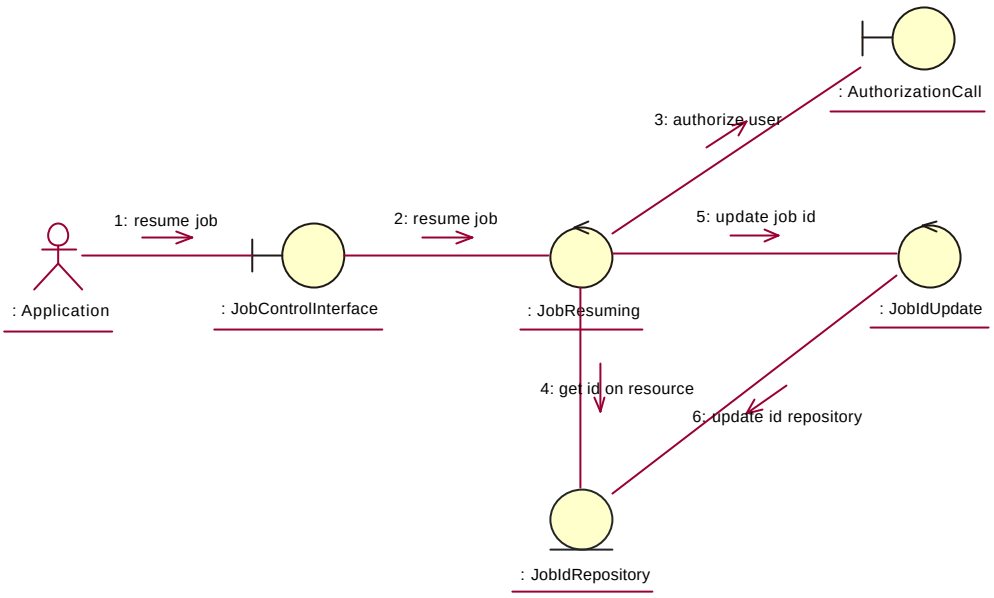
- **Remove Job**
  -Collaboration Diagram



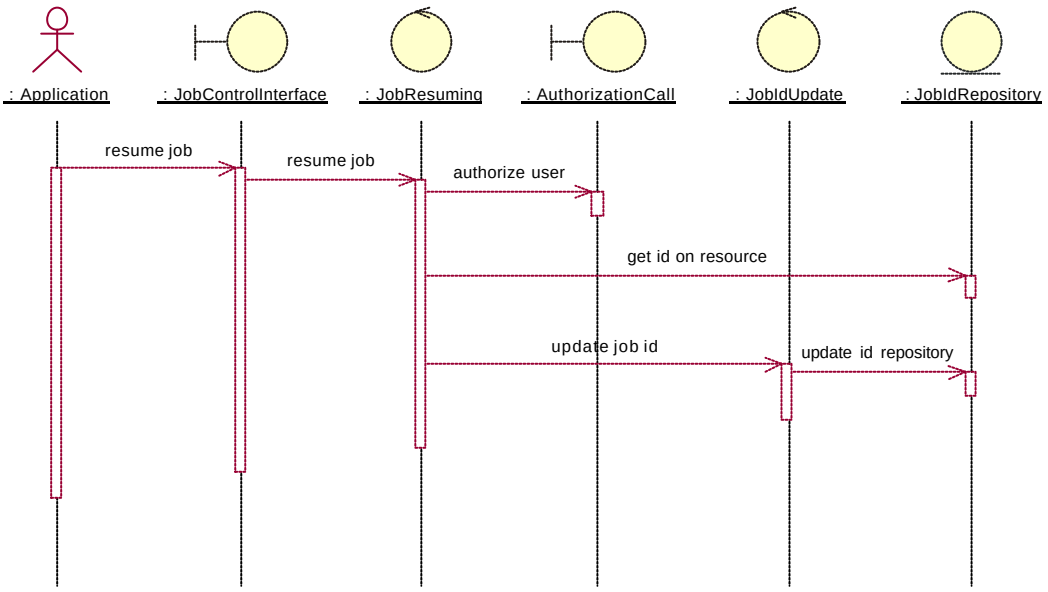
- Sequence Diagram



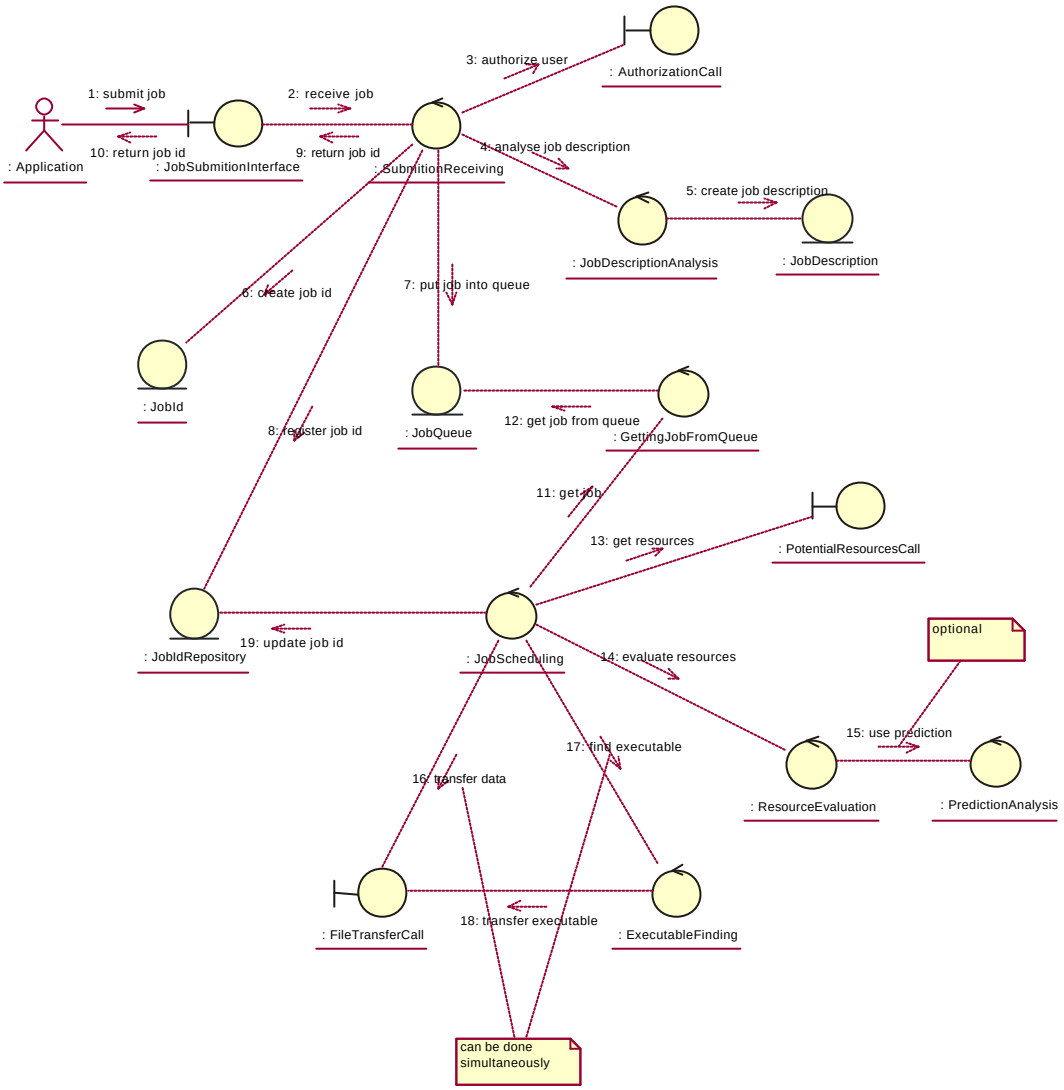
- **Resume Job**
  -Collaboration Diagram



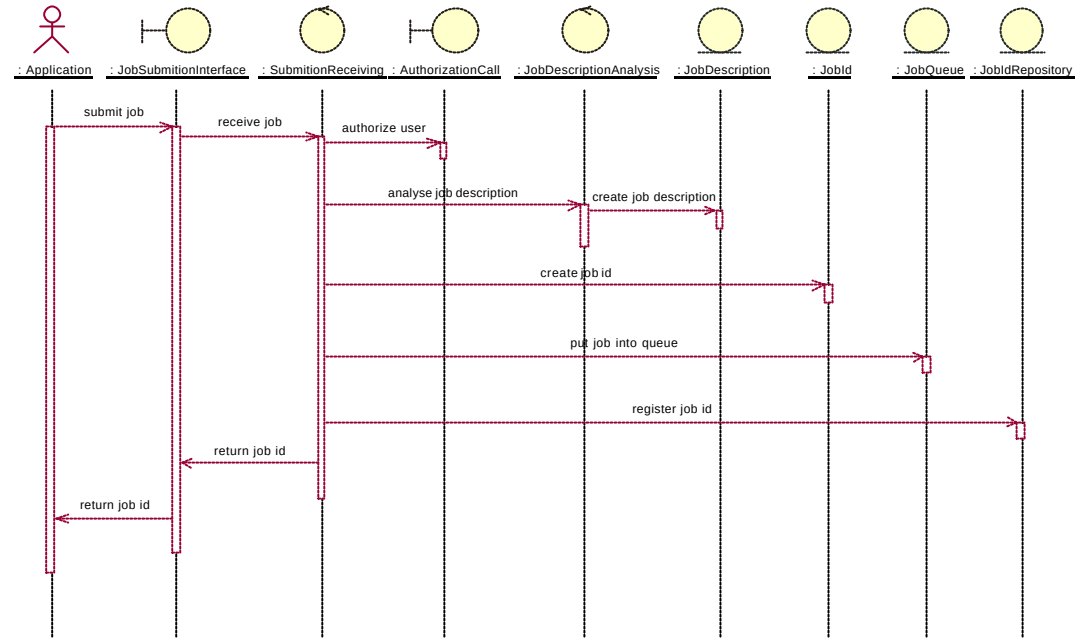
- Sequence Diagram

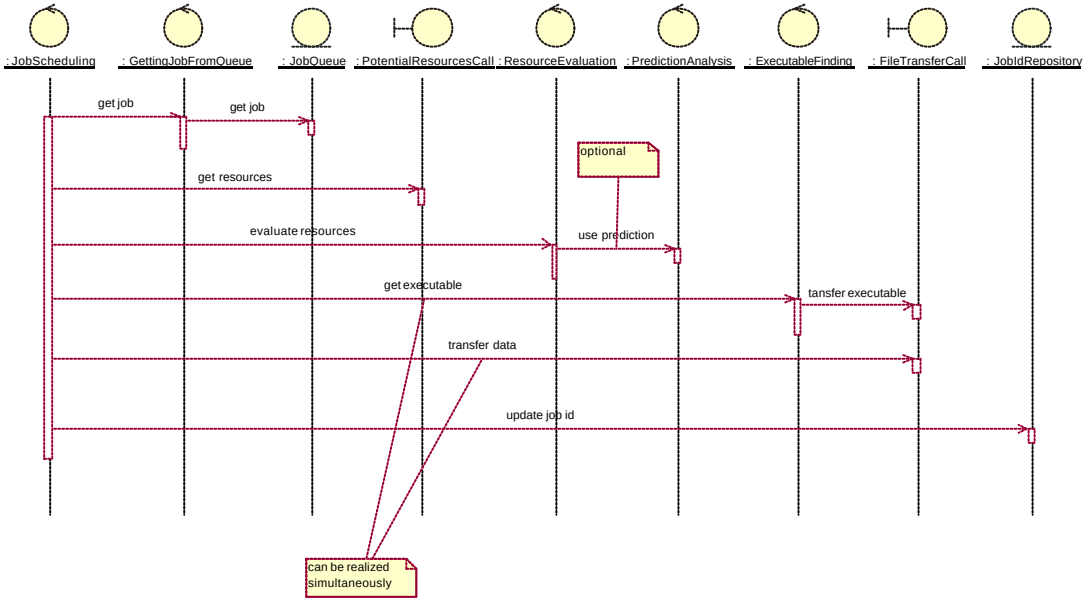


- **Submit Simple Job**
  -Collaboration Diagram

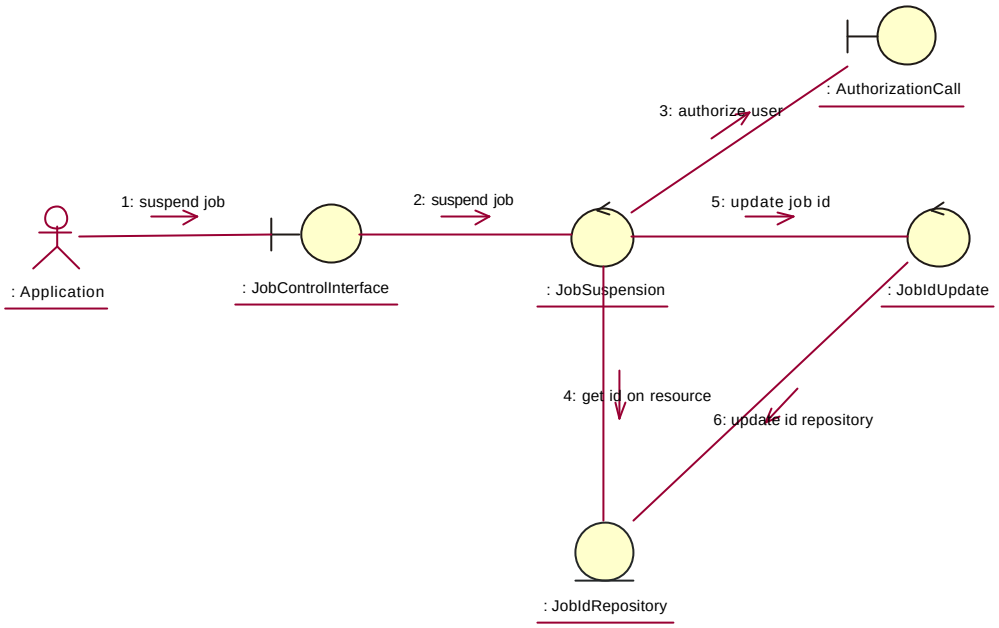


- Sequence Diagram

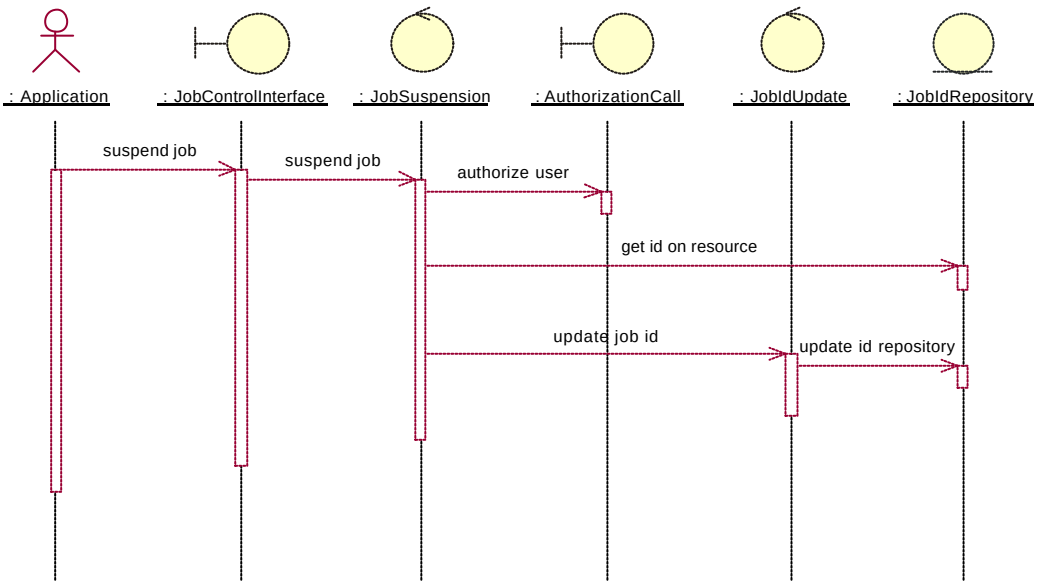




- **Suspend Job**
  -Collaboration Diagram



- Sequence Diagram



Deployment View

To be added in the future

Implementation View

In this section we show the pilot implementation of the GridLab Resource Management System, which have been developed for realization of the Migration Scenario from the GridLab Zakopane meeting. This Scenario has been presented in detail in D14.6 (Quarterly Report for the 3rd quarter). Here the general presentation of this scenario has been given and the pilot implementation of the GRMS has been presented.

Scenario

We have decided to develop and implement the scenario based on the idea of the job migration because of bad performance and using the GAT (Grid Application Toolkit) with all the underlying GridLab services. This scenario has been defined in the following way:

```
Job starting then Migration because of bad performance – final version
-----------------------------------------------------------------------

User starts job via portal, and gets notified when the job starts.

User then monitors GAT application through the portal and discovers that
it is performing badly.  The user requests migration to a system where
the application will perform better.

In the below security is implicitly involved in all communications.

(First stage - user asks portal to start job)

User <--> portal

  portal <--> MDS (where is resource manager)

  portal <--> resource manager

  resource management <--> MDS (list of resources)

  resource management <--> adaptive

 (resource management provides a list of candidate sites to the
 adaptation component)

 adaptive <--> monitoring system
```

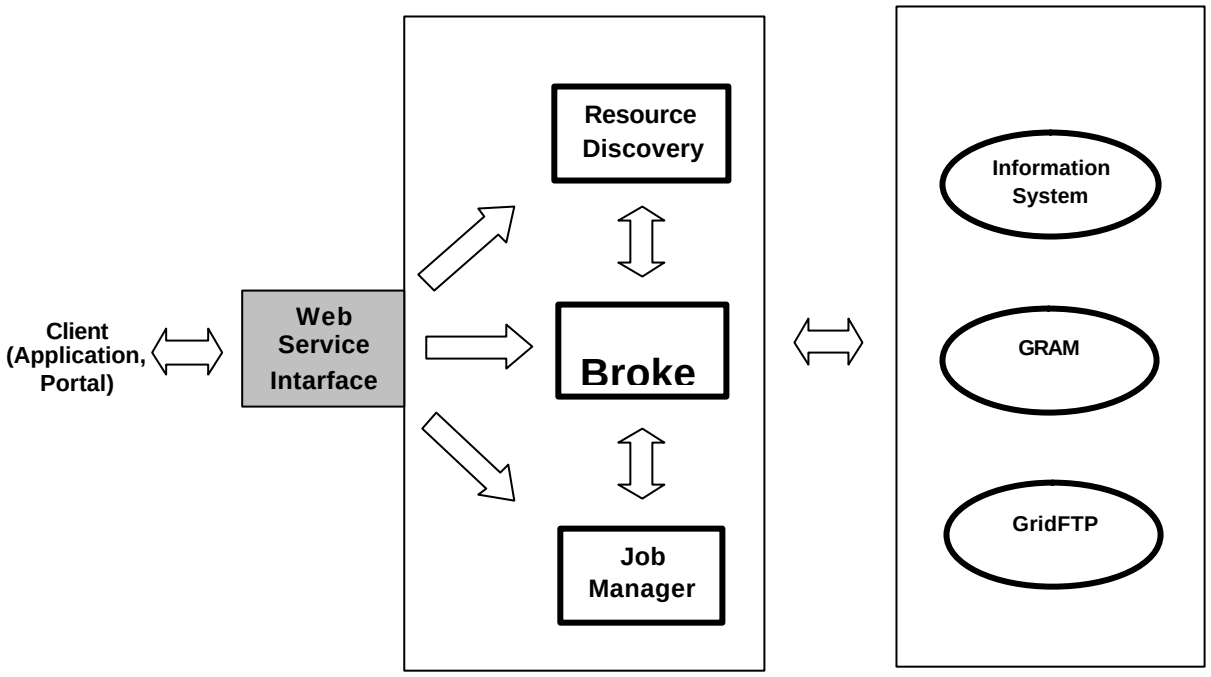
```
(adaptation component queries the monitoring system at each
candidate site about "execution time information".
This might be either: a "unified speed measure" a.k.a. BogoMIPS
                                - smallest possible precision
                                + but not that bad in practice
                        or: the result of an application-specific
                            micro benchmark
                            + quite expressive for the actual case
                            - application needs to provide this code
or: some prediction over previous runs of the same application
                            + might be the best thing to do
                            - this historical data needs to be stored
somewhere
                            - if there is no data, resort to BogoMIPS)

Adaptation component returns a ranked list of suitable candidate sites
to resource management

    resource management <--> replication (exe)

    replication  <--> file movement

    resource management <--> replication (input)

    replication  <--> file movement

    resource manager (submits job)

    resource manager <--> monitoring (here's a job to monitor with
gridlab id #1, local #2)

    (returns gridlab id #1)

    portal <--> resource manager  (where is monitoring service for #1)

    portal <--> monitoring service registers for notification event for
#1)

    monitoring service <--> portal (job #1 has started)

    portal <--> notification

    [application <--> monitoring (here is my process id #3, my gridlab
id is #1)]

(Second stage - user asks portal for performance info)

    User <--> portal

    portal <--> resource manager  (where is mon service for #1)

    portal <--> monitoring system (how is #1 performing)
```

```
    monitoring system <--> application

(At this stage the user has information about the job and decides the
job must be migrated)

(Third stage - user asks portal to migrate application)

    User <--> portal

    Portal <--> resource management (migrate job #1)

    resource management <--> MDS (list of resources, as above)

    resource management <--> adaptive

    adaptive <--> monitoring system

(At this stage the resource management system has found a new resource
to run the job on)

    resource management <--> application (checkpoint)

    application (checkpoints)

    application <--> replication (here are my checkpoint files)

    application (stops)

    resource management <--> replication (exe)

    replication  <--> file movement

    resource management <--> replication (input)

    replication  <--> file movement

    resource management <--> replication (checkpoint files)

    replication  <--> file movement

(All relevant data has now been moved to new hardware resource)

    resource management (submits job)
```

This scenario, including the mobile user support (notification) has been implemented using the Web Services technology. All the services and GAT are the pilot versions of the GridLab project. They are still being developed and the demo version will be presented at the SC2002 in Baltimore, 2002.

Overview

Figure below gives an overview of the pilot version of the GRMS system.



The pilot version of the system implements the following functionality:

- ability to choose the best resource for the job execution, according to Job Description and chosen mapping algorithm;
- ability to submit Simple Job according to provided Job Description;
- ability to migrate Simple Job to better resource, according to provided Job Description;
- ability to cancel job;
- provide information about job status;
- provide other information about job (name of host where the job is/was running, start time, finish time);
- provide list of candidate resources for job execution (according to provided Job Description);
- provide list of jobs submitted by given user;
- ability to transfer input and output files (gridFTP, GAAS);

The following components of the system have been developed and deployed:

- **Broker**: the most important component of the system responsible for:
  - Steering process of job submission;

- choosing the best resources for job execution (scheduling algorithm);
- transferring input and output files for job's executable;

- **Resource Discovery**: responsible for:
  - finding resources that fulfils requirements described in Job Description;
  - providing information about resources, required for job scheduling;
- **Job Manager**: simple module providing:
  - ability to check current status of job;
  - ability to cancel running job;

System interface

```
string submitJob(in string jobDescription);
```
Submits job according to jobDefinition; returns job id within Resource Management System

```
JobIdList getMyJobsList(in string username);
```
Returns list of job ids that were submitted by given user

```
void registerAppAccess(in string jobId, in string accessPoint, in long pid);
```
Registering callback access for running application (used to call application to checkpoint)

```
void migrateJob(in string jobId, in string jobDescription);
```
Migration of job with given job id, according to provided job description

```
string getHostName(in string jobId);
```
Returns name of host where job was executed

```
JobInformation getJobInfo(in string jobId);
```
Returns more information about job (status, machine, start time, finish time)

```
ResourcesList findResources(in string jobDescription);
```
Returns list of resources (resource manager contact strings) where described job can be executed

```
void jobCancel(in string jobId);
```
Cancel job witch given id

```
string getJobStatus(in string jobId);
```
Returns status of submitted job

Job Description

Job Description is a XML document which specifies:

- job executable:
  - file location,

- arguments,
- file argument (files which have to be present in working directory of running executable)
- environment variables
- standard input
- standard output
- standard error

- resource requirements of executable
  - name of host for job execution (if provided no scheduling algorithm is used)
  - operating system
  - required local resource management system (lsf, pbs, condor, etc.)
  - minimum memory required
  - minimum number of cpus required
  - minimum speed of cpu
  - there is some other parameter passed directly to GRAM (maxtime, maxwalltime, maxcputime)
- Job Description for migration can contain location of checkpoint files
- For the files location it is possible to use gridFTP and GASS urls.

Supported Job Description – general schema

```
< grmsjob appid = "value">
        <simplejob>
                <resource>
                 <osname> value </osname>
                 <hostname> value </hostname>
                 <localrmname> value </localrmname>
                 <memory> value </memory>
                 <cpucount> value</cpucount>
                 <cpuspeed> value </cpuspeed>
                 <maxtime> value </maxtime>
                 <maxwalltime> value </maxwalltime >
                 <maxcputime> value </maxcputime >
                </resource>

                <application>
                 <executable>
                         <url> value </url>
                 </executable>
                 <arguments>
                         <value> value </value>
                         …
                         <file type = "value">
                                 <name> value </name>
                                 <url> value </url>
                         </file>
                         …
                 </arguments>
                 <stdin>
                         <url> value </url>
                 </stdin>
                 <stdout>
                         <url> value </url>
                 </stdout>
                 <stderr>
                         <url> value </url>
                 </stderr>

                 <environment>
```

```
                    <variable name="value"> value </variable>
                    …
                <environment>
                <checkpoint>
                    <file type = value>
                        <name> value </name>
                        <url> value </url>
                    </file>
                    …
                </ checkpoint >


                </application >
        </simplejob>
</grmsjob >
```

Implementation of the pilot version

- Interface: GSI enabled web service based on Axis toolkit.
- System: components implemented in CORBA technology.

Notes

The pilot version of the GRMS system will be evaluated and still developed. As it is foreseen in the project schedule the full prototype version should be ready by the end of month 12. Then, the system will be documented in full.

Appendix 1 : Scheduling Algorithms for GRID Resource Management Systems

Scheduling in Grid Computing

Grid computing clusters a wide variety of geographically distributed computational resources, including supercomputers, PC's, PDA's and workstations, and presents them as a single unified integrated resource. [1]. Many computationally intensive problems can be solved within a more feasible/ reasonable time and cost based on a grid infrastructure than using a single resource supercomputing scheme [2]. Scheduling and resource management, which focus on optimizing Grid resource allocation, are important issues in implementing a computational Grid infrastructure. This is due to the fact that such an infrastructure can serve different jobs submitted by different users and therefore there is the need to determine when and on which processor competing tasks execute.

The number of users requested to be served and the jobs submitted vary from time to time in a dynamic framework. A job consists of a number of tasks, each of which is responsible for executing a certain process of the job. The tasks can be executed either independently one from the other or can be connected by dependencies which indicate temporal relations of the tasks [3]. In the latter case, we assume that a task, or a number of tasks should be first executed before other tasks start their execution. This is, for example, the case when a task awaits the results of another tasks. The tasks can be grouped into sub-jobs, each of which permits the task execution on a certain type of system or systems. For instance the some tasks require a given operation system for their execution.

Let us assume that in a Grid infrastructure, we have M available processors and N tasks requested to be served. The processors can be either identical or distinct with respect to function and speed. Then, the goal of a scheduler mechanism is to determine the way that tasks are assigned to the available resources. Each task T_i demands a number of instructions for its execution and thus a respective workload w_i if it is executed on a processor of unit capacity. Task workload is assumed to be a priori known to the scheduler, providing, for example, by a

prediction mechanism. In case that the workload w_i cannot be accurately predicted, we can assume that the task statistics are known. Particularly, supposing that the average workload $\overline{w}_i$ and the respective deviation s_i are available, we can estimate the worst execution time as $\overline{w}_i + g s_i$, where factor g determines the confidence we have about the task execution time.

Let us assume that the jth processor is characterized by capacity c_j. Then, assuming that a task occupies 100% of the processor utilization, its execution time is provided by

$$w_{i,j} = \frac{w_i}{c_j} \quad (1)$$

where $w_{i,j}$ is the workload of task i if it is executed on the jth processor.

The task execution is constrained by the task ready time $d_{i,j}$ and task deadline D_i. Ready time expresses the earliest time that the task T_i is available for processing on the jth processor, while deadline D_i the time beyond of which the task is prohibited to be executed. Ready time may results from communication delays or other networking constraints, while task deadline usually results from the users' requirements.

In the adopted scheduling schemes, the tasks are considered non-preemptable and non-interruptible. In particular,

- A task is said to be non-preemptable if once it starts its execution on a processor, it should complete its execution on this processor [3].
- A task is said to be non-interruptible if the processor to which the task has been assigned cannot interrupt the processing of the task to process another tasks [3].

Furthermore, we assume that the tasks are aperiodic, meaning that they do not arrive at periodic time instances, which is a reasonable assumption for the Grid environment.

Scheduling Evaluation and its Relation to the Charging Policy

In a scheduling problem the goal is to appropriately assign all the available tasks requested to be served to the available processors so that the time constraints, i.e., the task deadline and the task ready time are satisfied. In this way a feasible schedule is accomplished. Often, however, finding a single feasible schedule is not enough. In some scheduling problems, the goal is to find the optimal schedule among all feasible schedules, according to a desired optimality principle. This criterion is used to evaluate the scheduling performance. In additional, there are cases where a feasible solution cannot be reached, in the sense that some tasks cannot be scheduled to meet their respective time constraints. In this case, we also need criteria for evaluating the scheduling performance in order to select the most "appropriate" solution.

The measure used for evaluating a scheduling algorithm depends on the adopted system policy. In the sequel, the system policy depends on several techno-economical criteria and the charging policy used. Charging policy is an important issue in a Grid environment. For example, in case that some users have contributed more resources than others, it is fair the scheduler to favor tasks submitted by these users than the others. Other measure for evaluating the performance of a scheduler is to examine the occupied processor capacity. In this case, we assume that the charge is performed proportional to the time that a processor reserves. In the following, we present three different measures for evaluating a schedule algorithm and we also explain their relations with the charging policy used.

Success Ratio

A common measure for evaluating the scheduling performance is the *success ratio* of the tasks being feasibly scheduled (i.e., the tasks whose the respective time constraints are met) over the total number of tasks requesting for scheduling

$$E = \frac{\text{Number of tasks feasibly scheduled}}{\text{Total number of tasks}} \quad (2)$$

From an economic point of view, the previously described measure treats all tasks equally, regardless of the respective workload. Therefore, it is better to reject a high intensive task than to reject two or more low intensive tasks since only the number of feasibly scheduled tasks are taken into consideration. As a result, in this case the charging policy is performed according to the number of tasks being served.

Processor Capacity Occupation

Another measure is based on the workload reserved for all feasibly scheduled tasks. In this case, we have that

$$E = \sum_{\substack{\text{for all feasibly} \\ \text{scheduled tasks}}} w_{i,j} \quad (3)$$

According to this cost function, the scheduler service policy is provided per workload unit, meaning that it is more beneficial to serve tasks of demanded workload than tasks of small workload. This measure implies that the charging policy is performed with respect to the time that a processor reserves. This approach is closer to the charging policies used in common life, where a proportional charge to the customer demands (i.e., task workload) is adopted. This means that it is more preferable to serve few customers which are willing to pay a lot than a lot of customers which cannot pay a lot for their services.

Fair Scheduling Policy

Another charging policy is based on the number of resources that a user contributes to the Grid infrastructure. This means that tasks submitted by users with more contribution to the total resources should favor more than the other tasks. In section 8, we describe a new scheduling algorithm, which takes into consideration the user's contribution. The algorithm is based on the weighted Max-Min fair sharing scheme.

In particular, in case that the demanded task rate of a user is less than the respective weighting sharing rate, the task fair rate equals the task demanded rate. In this case, a remaining processor capacity is estimated, which is distributed to the tasks whose the demanded rates are greater than the initial sharing rates.

Earliest Deadline First

The purpose of a scheduler is to determine *when* and on *which* processor a given task executes. The word "when" refers to the order that the tasks should be assigned to the processors. For example, the third task, followed by the second task followed by the tenth task on a processor. Therefore, the problem is to select the most appropriate task among all the available unscheduled ones to be scheduled. The word "which" indicates at which processor the selected task is scheduled. In the following, we describe scheduling algorithms addressing both the aforementioned problems.

Perhaps the most widely used scheduling algorithm is the Earliest Deadline First (EDF) scheme also known as the deadline driven rule [4]. In particular, the method dictates that at any point the system must assign the highest priority to the task with the most imminent deadline. The concept behind the EDF scheme is that it is preferable to serve first the most urgent tasks (i.e., the task with the earliest deadline) and then serve the remaining tasks according to their urgency. Although, the EDF provides information about the order of the tasks that are to be scheduled (it answers the "when" question), it does not determine at which processor the selected task are assigned (it does not answer the "which" question). In the following, an Earliest Starting Time (EST) scheme are presented for addressing this issue.

Earliest Starting Time

In this approach, we assume that each task occupies 100% of the processor utilization for its execution. This means that each time only one task is executed on one processor. Therefore, in this scenario, the maximum utilization of a processor is assigned, implying the tasks are executed in the earliest possible time.

The processor at which a given task is assigned for serving is the one that provides the earliest starting time for the task. Let us denote as $r_{i,j}$ the starting time of task T_i served on the jth processor. Therefore, the most appropriate processor, say $\hat{j}$, is the one, which minimizes

$$\hat{j} = \arg\min_{j} r_{i,j} \tag{4}$$

The starting time $r_{i,j}$ depends a) on the respective ready time of the task (i.e., $r_{i,j} \geq d_{i,j}$) and b) the time at which the processor becomes available for executing the respective task. The latter condition stems from the task constraint, which says that no other task can share the resource at the same time (full utilization). Therefore, the starting time $r_{i,j}$ is provided by

$$r_{i,j} = \max(d_{i,j}, g_j) \tag{5}$$

where g_j the available time of the jth processor.

A simple approach for estimating g_j is estimated through the processor release time. The release time is the time at which all tasks scheduled on this processor have finished their execution. Figure 1 illustrates a scheduling scenario based on the EDF algorithm along with the EST approach, estimated by the processor released time. In this figure, we depict the ready time

of all scheduled tasks. Six tasks have been scheduled named 1,2,3,4,5 and 6 respectively onto two processors. The tasks have been ordered with respective to their deadlines.

Estimating the processor available time as the processor release time has the drawback that capacity gaps are created (see Figure 1), resulting in a waste of the processor capacity and thus in a deterioration of the scheduling performance. To overcome this problem, an alternative approach is adopted, which estimates the processor available time more precisely. Particularly, the capacity gaps are examined and in case that a selected task can be served within a capacity gap it is assigned to this time interval. Among all candidates time intervals the one, which provides the earliest starting time, is selected. Figure 2 presents how the scheduling of Figure 2 is improved by exploiting the capacity gaps. In this case, the completion time of tasks 5 and 6 is shorter than that obtained in Figure 2. Furthermore, the capacity of processor 2 is better exploiting, since the capacity gaps have been significant reduced. In addition, although the capacity gap of processor 1 is not reduced, its processing availability increases, leaving more capacity for scheduling forthcoming tasks.

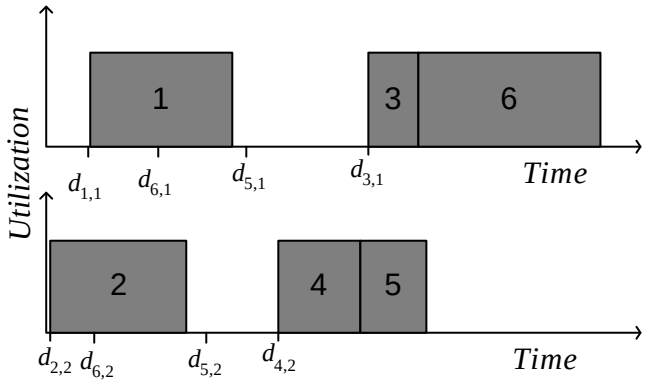


Figure 1. An example of the EST algorithm for processor selection with the available processor time estimated as the processor released time.

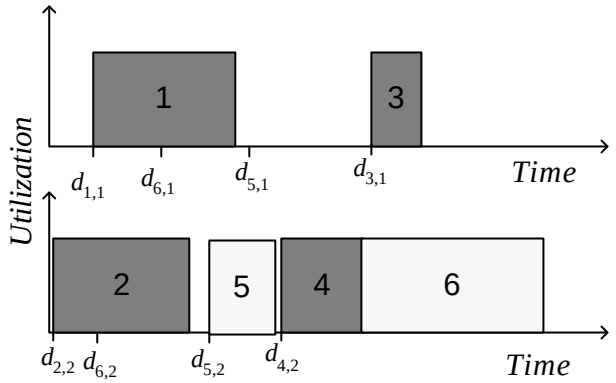


Figure 2 An example of the EST algorithm for processor selection by exploiting processor capacity gaps.

Least Laxity First

The least laxity algorithm also known as slack time algorithm dynamically assigns priorities to active tasks in order of not decreasing slack time, i.e., the difference between the task relative deadline and its remaining computational time [5], [6]. This least laxity algorithm measures the task relative urgency. Suppose that a particular task becomes active, while the processor is occupied with other work of higher priorities, preventing the task from receiving processing. The task initia l slack time is equal to the difference of the task deadline and the task completion time. However, as time passes, tasks slack time steadily decreases.

At this point, the processor must begin executing the task. If the processor does not begin executing the task and the slack time becomes negative, the task is sure its deadline. Using this idea, task selection is performed on their respective laxity values or slack time. The way that a task is assigned to a processor can be performed similarly to the previous case.

Branch and Bound Guided Searching

The previously described algorithms examine only one possible task arrangement for scheduling. This is due to the fact that tasks are ordered based their relative urgency (task deadline or laxity). It is possible, however, to increase the scheduling efficiency if other task arrangements are explored. For example, scheduling tasks with an order different than their relative urgency may improve the overall system feasibility. However, applying an exhaustive search to explore all possible combinations of task arrangements is practically impossible to be implemented due to large computational requirements, especially in case that a large number of

tasks and processors are available. For this reason, a guided searching algorithm is proposed for estimating a solution close to the optimal one of an acceptable computational complexity. In this report, two different guided searching methods are examined; The first is based on a branch and bound searching scheme and is described in the following, while the second on a genetic algorithm scheme and is discussed in section 7.

In the proposed branch and bound searching algorithm, a tree is initially created, each node of which corresponds to the task that is to be assigned to a processor, while the ancestor nodes to the already scheduled tasks. Again, non-interruptible and non-preemptable tasks are assumed. We also assume that each task occupies full utilization of the processor capacity for its execution. The algorithm starts from the root of the tree, which is an empty schedule, and tries to extend the schedule with one more task by moving to one of the vertices at the next level in the search tree until a full feasible schedule is derived. If the current vertex is a non-feasible solution, meaning that the selected task cannot be feasibly assigned to a processor, the algorithm backtracks to the previous search point so that another possible path is examined.

At each node of the tree, the most appropriate task is selected for scheduling among all the unscheduled ones. As the most appropriate task, we select the one, which provides the greatest probability of being the next schedule feasible. For this reason, a heuristic function, say *H*, can be used to check the feasibility performance of the remaining task and the task with minimum value of *H*, is selected as the most appropriate

$$T_l = \arg\min_{l \in U} H \tag{6}$$

where refers to the set of unscheduled tasks.

Several heuristic functions can be used as *H*. A common way is function *H* to be equal to the task deadline or task laxity, meaning that the most probable task for selection is the ones of relative urgency. Another interesting selection for function *H*, is based on the combination of the task deadline and the task earliest starting time [7], [8],. Particularly,

$$H(T_i) = D_i + W \cdot \min_{j \in P}(r_{i,j}) \tag{7}$$

where W is a constant which regulates the importance of task deadline D_i compared to the task starting time $r_{i,j}$. Large values of W indicate that the minimization is mainly affected by the deadline. On the contrary, small values of *W* mean that the minimization is primarily affected by the starting time.

To improve the computational efficiency of the algorithm, the feasibility performance is checked on a predetermined small number of unscheduled tasks, say *K*, i.e., over a feasibility check window, instead of examining the all the unscheduled task. If the value of *K* is constant (and in practice *K* is smaller than the task set size *N*), the complexity of the algorithm is linearly proportional to *N*, i.e., the number of tasks [8], [9]. The main steps of the proposed scheduling algorithm are summarized in Table I.

1. Order the tasks in non-decreasing order of deadline in the task queue. 2. Start with an empty schedule. 3. Select a task by minimizing function *H* (Equation (7)) 4. If the selected task is found to be feasible, (a) Set the task in the scheduled set (b) Estimate the task starting time and the processor that the task is to be served using the EST method. 5. else (a) Backtrack to the previous search level. (b) Extend the schedule with the task having next best *H* value. 6. Repeat steps (3-5) until a termination condition is met.

Table I: The main steps of the branch and bound scheduling algorithm (case 1)

Figure 3 shows a tree created in case of 4 tasks. The figure illustrates the possible paths examined by algorithm. The grayscale node corresponds to the obtained feasible solution.

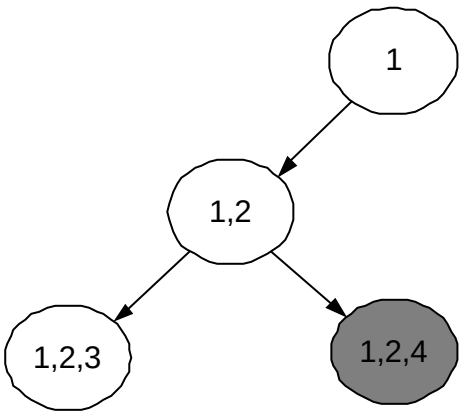


Figure 3. An example of the branch and bound guided searching scheme in case of 4 tasks requesting for scheduling

The algorithm terminates either when a) a complete feasible schedule is found or b) a maximum number of backtracks is reached, or c) when no more backtracks are possible.

Uniform Processor Utilization

In the previously described algorithm, all tasks are served using an 100% utilization degree of the processor capacity. As a result, the completion of a task execution time is as small as possible. However, in real life systems, tasks dynamically arrive and the scheduling algorithm is repeatedly activated to serve the new incoming tasks. In this case, if initially the scheduler finds a feasible solution by allocating 100% of the processors' capacity, it is more difficulty at the following scheduling activation to find a feasible solution for the new arriving tasks, especially in case that the initially scheduled tasks are characterized by long execution times. This scenario, is depicted in Figure 4. Particularly, initially the scheduler has assigned two tasks on a processor for execution. At time *t*, a new scheduling process is activated to feasibly serve the new incoming tasks. In this scenario, we assume that one new task has arrived. The deadline of this task is also presented in Figure 4. As can be seen, the deadline of the new forthcoming task (task 3) is such that this task cannot be feasibly served on the processor, since the already scheduled tasks cannot be interrupted. In this case, it would be more preferable the scheduler to initially assign the task with a low utilization degree, exploiting as much as possible the task deadlines, leaving available processor capacity for the new incoming tasks. This is presented in Figure 4, where task 1 and 2 have been scheduled with utilization degree of 25% and therefore task 3 can be feasibly served using a utilization degree of 50%.

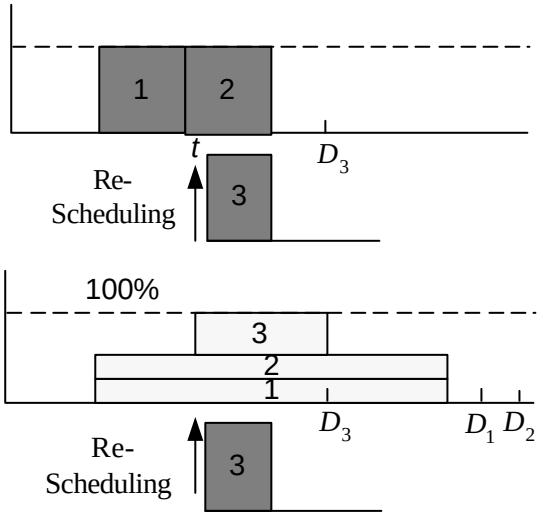


Figure 4. A scenario depicts that a utilization of 100% can lead of violation of the time constraints for forthcoming tasks.

Furthermore, another drawback of the 100% utilization approach is that the tasks are not handled in a fair manner. For example, let us suppose that two tasks with the same deadline are assigned to a processor in a feasible way. In this case, the first executed task finishes its execution much earlier than the second task which is not fair policy. This scenario is depicted in Figure 4, where the tasks 1 and 2 present almost the same deadline but different completion times.

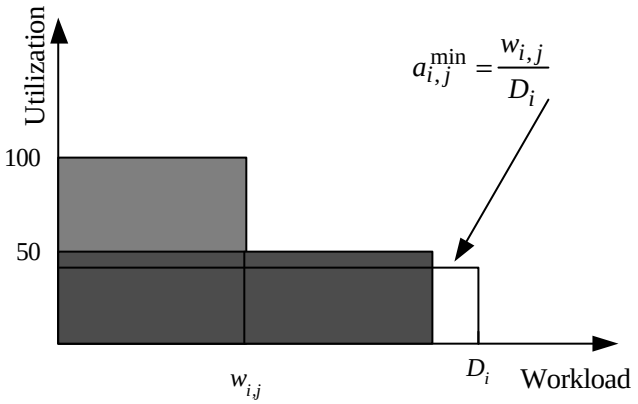


Figure 5. Variation of the task execution time with respect to the utilization degree.

To overcome the aforementioned difficulties, a different scheduling approach can be adopted by assigning a lower than 100% utilization degree for each task execution. The task execution time depends on the assigned utilization degree. This is illustrated more clearly in Figure 5, where different utilization degrees have been used for the task execution. Since the task execution is constraint by its ready time and deadline, a minimum utilization degree can be estimated for each task as follows

$$a_{i,j}^{\min} = \frac{w_{i,j}}{D_i - d_{i,j}} \tag{8}$$

One solution is to serve each task by assigning the lowest possible utilization degree as provided by the previous equation. However, since the tasks are non-preemptable such an approach cannot be always valid in practice. This is clearly illustrated in Figure 6. In this figure, we assume that initially two tasks have been scheduled by assigning to each the lowest possible

utilization degree, as defined in equation (8) so that the deadlines D_1 and D_2 are satisfied. As can be seen from Figure 6, for both tasks T_1 and T_2, the lowest utilization degree is 50%. Let us now assume that a third task T_3 is to be scheduled, with a minimum utilization of 25%. It is clear that the third task cannot be spread in the interval $[d_3\ D_3]$ in a non-interruptible way.

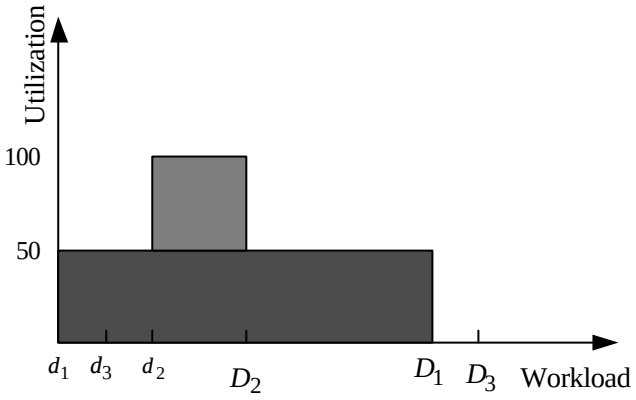


Figure 6. An example of the utilization profile created by assigning tasks of different utilization degree.

Estimation of the Utilization degree

To estimate the utilization degree assigned for a task, we need to know the utilization profile of the processors. The utilization profile is formed by the tasks, which have been already scheduled to a processor. Let us denote as $g(t)$ this utilization profile. Since each task is served with a constant utilization degree throughout its execution, function $g(t)$ presents a step wise form. Let us assume that $g(t)$ has zero value for $t>t_N$. Then if $0=t_0<t_1<\cdots<t_N$ are the points at which $g(t)$ changes value, due to either the completion or the starting of a task execution, function $g(t)$ is defined as follows

$$g(t)=\begin{cases} b_i & \text{if } t_i \le t < t_{i+1} \\ 0 & \text{if } t \ge t_N \end{cases} \qquad (9)$$

Using the utilization profile $g(t)$, we can find a possible time interval that a new incoming task can be executed. Since the task T_l requires at least $a_l^{\min}$ utilization to be executed intervals with utilization $b_i + a_l^{\min} > 1$ cannot satisfy the T_l requirements and thus should be excluded for searching. Similarly, time instances lower than the task ready time $d_{i,j}$ and greater than

D_i should be excluded. The remaining intervals are possible intervals for scheduling task T_l. In this way, an indication function is formed which takes values 0 and 1. Zero values correspond to intervals on which a selected task cannot be feasibly scheduled. On the contrary, values of 1 correspond to possible time intervals, which can be feasibly executing a selected task. This is illustrated in Figure 7 for the example of Figure 6.

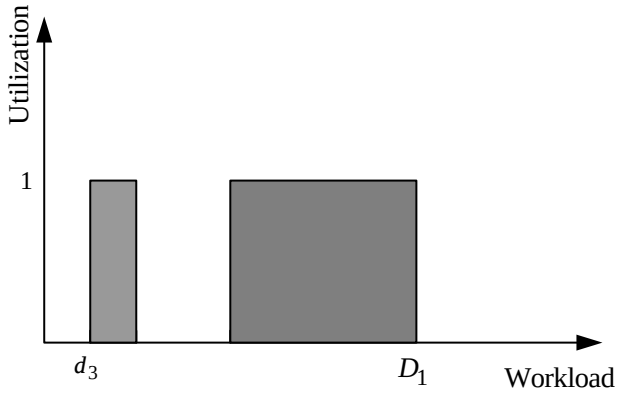


Figure 7. The indicator function for the utilization profile of Figure 6.

The interval, which is selected for executing the task is the one that a) can serve the task feasibly and b) retains the utilization profile as low as possible. Let us denote as l an examined time interval. Then, the processor capacity characterizing this interval is defined as follows

$$C(l) = \max g(t),\ t \in l \tag{10}$$

In addition, the minimum utilization degree $a(l)$, which should be assigned for the task execution requires in the interval l is provided as

$$a(l) = \frac{w_i}{len(l)} \tag{11}$$

In case that $a(l) + C(l) \le 1$, the task can be feasibly executed in the interval l. Among all possible time intervals which yields feasible task execution, the one which retains the lowest utilization profile is selected as the most appropriate.

$$\hat{l} = \arg\min_{l \in L} \{C(l) + a(l)\} \tag{12}$$

where L denotes all possible tasks for executing the task.

The set L contains all possible intervals on which the indication function is 1. Other possible intervals can be obtained by decomposing these intervals into partitions as explained graphically in Figure 8. Particular, Figure 8(a) a utilization profile is presented, in which four tasks have been served. The possible intervals that a new task with ready time δ and deadline D can be served are shown in Figure 8(b). The new task can be assigned in these intervals, if this is possible or in any combination of these intervals.

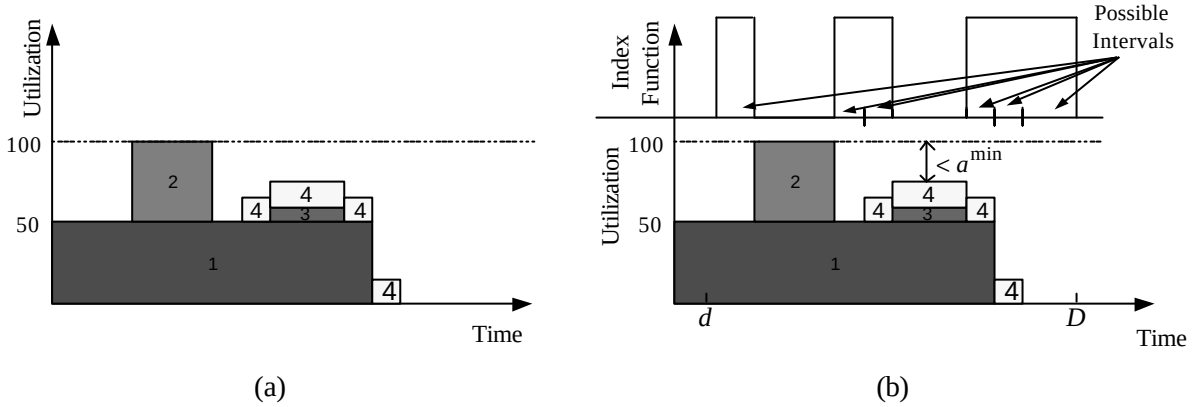


Figure 8. Possible time intervals that a new task can be assigned.

Similarly, we can estimate the most appropriate processor for scheduling, i.e., the processor, which yields the lowest utilization profile among all processors.

It should be mentioned that the proposed scheme estimates the appropriate processor that a selected task should be executed and the respective utilization degree used for its execution. The order that a task is selected for scheduling is based either on a relative urgency scheme, such that the EDF or on a branch and bound scheme and a genetic algorithm scheme as described in the following.

Genetic Algorithm

In the previous section we describe a branch and bound searching scheme for task order estimation. An alternative approach is used a genetic algorithm scheme. In this case, possible solutions for scheduling are represented as chromosomes whose “genetic material” corresponds to a specific task arrangement on the available processors [10]. In order to apply a genetic algorithm, we need to determine the following: The representation scheme used to encode the genes as tasks'

arrangements, the adopted crossover operator, the initial population and the mutation scheme, which introduces random gene variations that are useful for exploring new search areas.

Chromosome representation

The chromosome representation scheme describes the way that the N available tasks are arranged onto the M processors. In this framework, if a chromosome is known, it is also known how the N tasks are scheduled. In our case, a matrix $\mathbf{X}$ is used for representing the chromosomes. The elements of $\mathbf{X}$, say $x_{i,j}$, take value from 1 to N, i.e., $x_{i,j} \in \{1,....,N\}$ each of which corresponds to a given task, say T_i, among all N available. The index j of $x_{i,j}$ indicates the processors at which $x_{i,j}$ is assigned to be scheduled, while index i corresponds to the order that the task is to be scheduled. For example, $x_{2,4}$ represents the second task scheduled on the fourth processor. Figure 9 presents an example of the proposed chromosome representation in case of M=2 and N=5. In this particularly example, tasks 2, 5, and 1 are assigned to first processor, while the tasks 4 and 3 to the second processor.

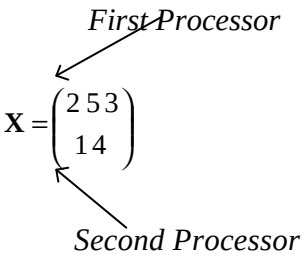


Figure 9. An example of the proposed chromosome representation. Five tasks are assigned to two processors. Particularly, tasks 2, 5, 3 are assigned to the first proc essor, while tasks 4 and 3 to the second processor.

Initialization of the Population

The genetic algorithm is initialized based on any of the aforementioned described scheduled schemes, such as the Earliest Deadline First (EDF) algorithm [4], combined with an appropriate processor allocation scheme such as the earliest starting time. Based on the aforementioned statements, an initially chromosome is generated, denoted as $\mathbf{X}(0)$. The initial population affects the number of iterations required for the genetic scheme to be converge to the optimal solution. An initial population which is close to the optimal scheme in general demands much less

iterations of the genetic algorithm than a randomly selected initial chromosomes. This is due to the fact that it is more preferable to start from solution close to the optimal one than from a random one.

Crossover Operator

The crossover operator indicates the way that the genes of a chromosome are exchanged in order to produce a new chromosomes. Using the crossover operator and the mutation mechanism, we move from one possible solution (task arrangement) to another possible solution and thus from one iteration of the algorithm to another one. The goal of the crossover mechanism is to perform a task re-arrangement (gene exchange) so that it is more probable a better solution is obtained [11]. For this reason, we move unscheduled tasks to positions which present higher probability of being feasibly scheduled. In the following, two different scenarios are considered. The first assumes that tasks explore 100% of processor utilization, while the second that a different utilization degree can be assigned for each task.

Full Utilization

Let us assume that an unscheduled task T_l is selected for re-arrangement, with a ready time $d_{l,j}$ and a deadline D_l. This task is positioned in another location so that a higher probability of being feasibly scheduled is encountered. Tasks are permitted to be executed in the interval $[d_{l,j}\ D_l]$. Therefore, it is expected that only the feasible tasks whose the execution time located in $[d_{l,j}\ D_l]$ affect the T_l execution and these tasks are considered possible for re-arrangement. A probability of selecting a feasible task to be substituted for the unscheduled task T_l is related to the laxity of the completion time of the task and the deadline of the unscheduled task undergone re-arrangement. This value is defined as

$$d = D_l - r_T - w_T \tag{13}$$

where r is the starting time of the selected feasible task T, while w_T corresponds to the workload of the task T. Particularly, the lower the value of d is the more probable is the respective task to be selected. This means that we move the unscheduled task to the less feasible position with a probability value.

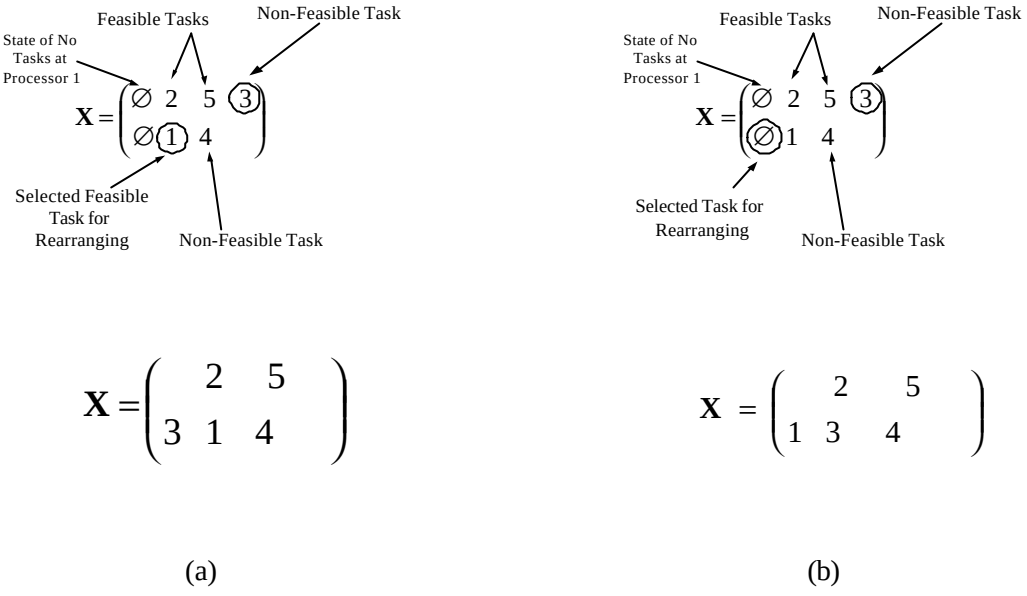


Figure 10. An example of the proposed crossover operator. a) Feasible tasks are possible for re-arranging the selected unscheduled task (3). Task (1) is selected for re-arranging. b) The new chromosome obtained after task re-arrangement.

If no tasks are executed in the interval $[d_{l,j}\ D_l]$, d equals to the deadline of the D_l of the selected unscheduled task. An example of the proposed gene exchange is in Figure 10. Tasks illustrated as ∅ corresponds to a dummy task so as to examine the case where no tasks are in the interval $[d_{l,j}\ D_l]$. In this case, we assume that all tasks are in this interval. In the first scenario, the non-feasible task 3 is moved in the right left position of the feasibly scheduled task 1 so that a new arrangement is generated. The task re-arrangement is presented in Figure 10. In the second scenario, the selected task is positioning in the first order of the two processor, since the dummy task is selected.

No Full utilization

In case that a different degree of utilization can be assigned to each task, a similar crossover operator is adopted. As in the previous case, it is expected that only the feasible tasks whose execution time starts or finishes within the interval $[d_{l,j}\ D_l]$ defined by the unscheduled task T_l

are examined. In this case, however, a different function is adopted to determine the probabilities assigned for all feasible candidate task. In particular, in this case the probability is inverse proportional to the utilization degree. This means that a task with a low utilization is more probable to be selected compared to a task of high utilization.

Another possible scenario is to serve the unscheduled task as the same position and processor, but using a different utilization degree. This is also performed in our case according to a probability value say p_u. In particular, if a random number $r \in [0,1]$ is greater than $r > p_u$, only the utilization degree assigned for the task T_l changes if this is possible.

Fitness Function and Mutation

The next step of the genetic algorithm is to apply *mutation* to the newly created chromosomes, introducing random gene variations that are useful for restoring lost genetic material, or for producing new material that corresponds to new search areas. *Uniform mutation* is the most common mutation operators and is selected for our optimization problem. In particular, each offspring gene $x_{i,j}$ is randomly position to any location with a probability value p_m. That is, a random number $r \in [0,1]$ is generated for each gene and replacement takes place if $r > p_m$; otherwise the gene remains intact.

The performance of a population is evaluated using the measures defined in section 2. Several GA cycles take place by repeating the procedures of fitness evaluation, parent selection, crossover and mutation, until the population converges to an optimal solution. The GA terminates when the best chromosome fitness remains constant for a large number of generations, indicating that further optimization is unlikely.

Fair Scheduling

In this section, we face the scheduling problem from a different point of view by handling the scheduling as an admission control scheme. In particular, the aforementioned described approaches a) selects an appropriate task for scheduling among all the unscheduled ones and b) estimates an appropriate time interval for executing this task on a suitable resource so that the respective time constraints, i.e., the ready time and deadlines, of the given task are met. Instead, in this alternative approach, the scheduling is performed based on the demanded task rate, which is defined as the fraction of the task workload over the time that the task is permitted to be executed.

$$X_i = \frac{w_i}{D_i - d_i} \tag{14}$$

In equation (14), d_i is defined as the weighted average of ready times $d_{i,j}$ of the ith task over all available processor, so that the demanded task rate is independent from the processor that the task is assigned to.

$$d_i = \frac{\sum_j d_{i,j} C_j}{\sum_j C_j} \tag{15}$$

where C_j refers to the processor capacity.

The X_i express the rate that the processor should allocate for executing the examined task and is independent of the way that the task T_i is processed on the resources. For example, the task can be executed with different utilization degrees but the total rate is equal to the demanded one. Therefore, in this case, the scheduling scheme instead of determining appropriate intervals for task execution, it defines to which resource the task is assigned and at which rate. Such an approach leads to a fair scheduling policy of the requested tasks. A fair scheduling is related on the number of resources that a user contributes to the Grid infrastructure. This means that tasks submitted by users with more contribution to the total resources should favor than the remaining tasks. In particular, let us suppose that a processor capacity overflow is accomplished, meaning that the total demanded rate of the tasks assigned to this processor is greater than the processor capacity. This means that the rates of some or all tasks in this processor should be reduced.

Assuming that all users equally contributes to the Grid infrastructure, a fair scheduling policy is to equally reduced the rates of all assigned tasks with respect to the capacity overflow. On the contrary, applying any of the aforementioned scheduling schemes some tasks are to be scheduled non-feasibly in favor of the feasibly scheduled tasks.

Problem Formulation

Let us also assume that P processors are available and each processor is characterized by a capacity, say C_i. Our goal is to assign the tasks T_i, $i=1,2,\ldots,N$ to the P available processors so that the tasks demanded rates are reached as much as possible. However, this requirement cannot be always satisfied. This is for example the case when the sum of demanded rates are greater than the total capacity offered by the processors, i.e.,

$$\sum_{i=1}^{N} X_i > \sum_{j=1}^{P} C_j \qquad (16)$$

Furthermore, even in case that the overall processor capacity is greater than the total demanded rate it is probable some tasks to be executed with a rate lower than the required one. This is due to the fact that in our case we assume that a task is entirely served by one processor. This can be seen in Figure 11, in which two processors are considered with capacities 35 and 50 respectively. In this example, we assume that 5 tasks are to be scheduled with rates, 25, 25, 15, 15, and 10 respectively.

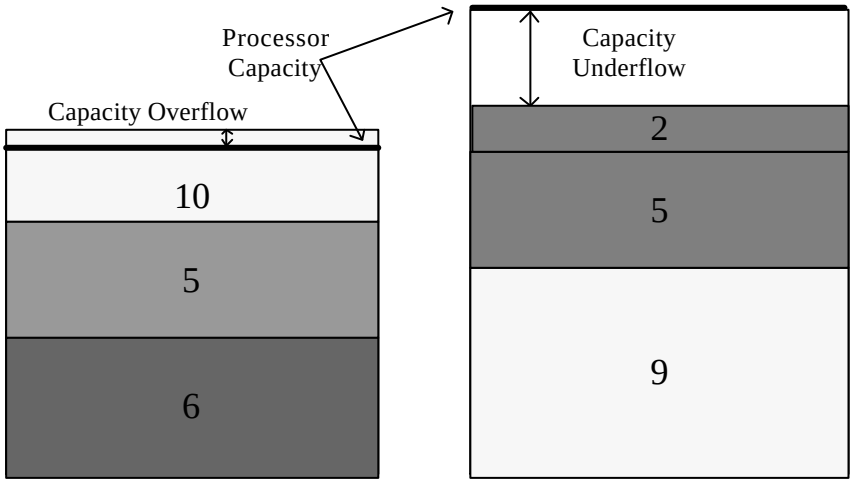


Figure 11. The idea of the fair scheduling.

In case that the demanded rates cannot be fulfilled, reduction of the task rates is accomplished so that the processor capacity constraints are satisfied. In our case, the tasks are modified using a fair policy obtained by the Max-Min fair sharing algorithm [12]. Let us denote as $\hat{r}_i$ the fair task rates. These rates are derived by fairly sharing the processor capacity to the available task according to the contribution of the users to the Grid resources. An example of the Max-Min fair sharing algorithm is shown in Table II. In this table, apart from the demanded and fair rates of the tasks, we also present the weights corresponding to the task contribution to the total recourses. Tasks, whose the demanded rates X_i are less than the initial capacity rate sharing to this task, are served with the demanded rates. The remaining capacity is sharing to unsatisfied tasks according to their contribution to the total resources.

Demanded Rates	Weights	First Weighted Sharing Rates	Fair Rates
10	1	5	5.66
8	2	10	8
5	1	5	5
15	2	10	10.66

Table II. An example of the Max-Min fair Sharing algorithm if the overall processor capacity is 30.

The Max-Min fair sharing algorithm can be successfully applied for task scheduling in case that the rate of a task can be split in any piece and then process by any one of the available processors. In our case, however, we assume that the rate of one task can be only served by one processor. As a result, it is probable the scheduled task rates, say r_i, are different of the rates $\hat{r}_i$, even in case that equation (15) is not satisfied.

Therefore, our goal is to estimate the scheduled task rates r_i so that they are as close as possible to the fair task rates $\hat{r}_i$ and simultaneously the processor capacity constraints are not violated under the condition that each task is scheduled only to one processor. These conditions are expressed in the following equations

$$\min \sum_{i=1}^{N} \left| \hat{r}_i - r_i \right| \quad \text{or} \quad \min \max_i \left| \hat{r}_i - r_i \right| \tag{17a}$$

$$\sum_{i \in B_j} r_i \le C_j \qquad B_j = \{ i : T_i \text{ scheduled on } j \text{ processor} \} \tag{17b}$$

$$r_i \le \hat{r}_i, \text{ for all } i \tag{17c}$$

Equation (17b) means that, the scheduled rates are restricted by the capacity constraints. The constraint imposed by equation (17c) indicates that the scheduled rates should be smaller or equal to the fair rates. Initially, the scheduler assigns the tasks using as rates the fair ones $\hat{r}_i$ so that equation (17a) is minimized. In case that equation (17b) is not valid, the scheduled rates should be reduced to satisfy the constraint (17b). Otherwise, there is a remaining capacity on the jth processor and, since the error of (17a) is zero over this processor, there is no reason to increase task rates beyond fair rate $\hat{r}_i$.

Task Re-arrangement

Minimization of equation (17) is a computationally intensive problem, which resembles to the bin-packing algorithm. This means that the optimal solution can be found if all possible combinations of the task arrangement should be examined. However, this is practically impossible especially for large number of tasks and processors. In the following, we propose a task re-arrangement scheme of polynomial complexity, so that the total error expressed either by the first or the second term of (17a) is reduced.

The idea behind the proposed task re-arrangement scheme is to efficiently combine processors overflows with processor underflow so that a better exploitation of the overall processor capacity is accomplished. In particular, let us denote as O_j and U_k a processor overflow and underflow respectively,

$$O_j = \sum_{i \in B_j} \hat{r}_i - C_j, \quad j : O_j > 0 \tag{18a}$$

$$U_k = \sum_{i \in B_k} \hat{r}_i - C_k, \quad k : U_k < 0 \tag{18b}$$

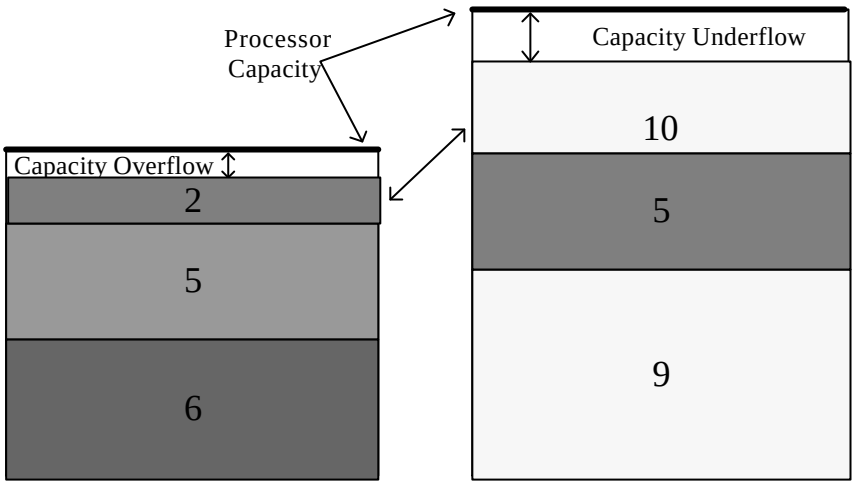


Figure 12. Task Re-arrangement for improving the processor overflow.

Let us also assume that we substitute a task rate r_l of the overflowed processor with a rate r_k of the undeflowed processor. It is clear that, after the task re-arrangement, the processor overflow and underflow are updating as follows

$$O'_j = O_j - c \tag{19a}$$

$$U'_k = U_k - c \tag{19b}$$

where $c = \hat{r}_m - \hat{r}_l$ expresses the task rate difference and O'_j and U'_k the updating processor residual. Taking into consideration equation (19), we can estimate appropriate bounds for the task rate difference c so that, after the re-arrangement, reduction either of the first term or the second term of (17a) is accomplished. This is depicted in Figure 12, where the task of rate 10 is substitute for the task of rate 12. After substitution no processor overflow is encountered.

In case that the second term of (17a) is used for the minimization, the overall performance can be improved even if two or more overflowed processors are combined. In this case, we try to equalize the overflow errors between the processors. The error reduction is performed in a similar way.

Initial Task Arrangement

For implementing the aforementioned described scheme, an initial task arrangement is required. This is addressed is this section. Several heuristic schemes can be applied to perform the initial task arrangement. In our case, a method is adopted which yields in most times a good initial task arrangement.

Particularly, the algorithm arranges the tasks with respect to their rate. In particular, the task of the greatest rate is initially scheduled followed by the second greater task and so on. The tasks are assigned to a processor so that a) they can feasibly scheduled and b) the best fit the processor capacity. The first condition means that the available processor capacity is enough to schedule the task. The second condition indicates the residual capacity, after task arrangement, should be as low as possible. In case that a selected task cannot be feasibly scheduled on any processor, the task is assigned to the processor on which a minimal overflow is encountered. The processor overflow is measured using equation (18). This process is terminated until all tasks are scheduled on the processors.

Assuming that a large number of tasks request to be scheduled, the proposed scheme tries to equalize the residuals of the processor capacity. Therefore, it tries to minimize the error expressed by the second term of (17a). As a result, the algorithm assigns the tasks towards a fair direction.

Fair Sharing of Capacity Overflow

In case that the tasks rates scheduled on a processor are greater than the processor capacity, reduction of the task rates should be performed. The way that the task rates are reduced depends on the adopted scheduling policy.

The function used for describing how close are the scheduled rates to the fair rates (the adopted norm) defines the adopted scheduling policy. In particular, let us assume that the scheduled rates are estimated so that the overall processor overflow (error) is minimized. This is expressed by the first term of equation (17a). Then, the overflow can be randomly distributed to over the respective processor tasks. For example, in this approach it is equivalent to assign the overall error to one task (if this is possible) or to equal sharing the error over all tasks of the respective processor. It is clear that such an approach does not yield a fair scheduling policy.

On the other hand, using the second term of equation (17a) a fair scheduling policy can be obtained. This is due to the fact that, in this case, the error for each individual task should be as low as possible. Particularly, let us denote as g_i the weights of the tasks, which express the user

contribution, which submits the tasks, to the Grid resources. It is anticipated that tasks submitted by users, which contribute more to the task resources, should favor against the remaining tasks. It can be found that, for a given task arrangement, if the rate of all tasks scheduled on the jth processor is reduced by the quantity

$$\hat{e}_{i,j} = \frac{O_j}{n_j \cdot g_i} \tag{19}$$

then, the second term of equation (17a) is minimized. In equation (19), n_j expresses the number of tasks on the jth processor. In case that all users are equally contribute to the system resources (i.e., g_i=1), the processor overflow is equally sharing to all tasks scheduled on this processor.

Task Order Estimation

The aforementioned algorithm estimates at which processor a task is served and at which rate. However, it provides no indication about the order of the task execution on a processor. This can be performed using a method similar to that used in the GPRS algorithm [13].

Particularly, according to the tasks scheduled rates, we estimate the completion time of the tasks on a processor, assuming that all are scheduled using a fair policy. Then, the task order is estimated based on the order of the task completion time. This means that the task with the minimum completion time is first executed, followed by the next task and so on.

Dependent Tasks

In the previous sections, we have assumed that the tasks are independent one from the other. There are cases, however, in which the tasks should be connected by dependencies. This means that some tasks should wait the execution of other tasks before their execution.

To handle this case, a graph can be constructed, each node of which corresponds to a task. The ancestors of the nodes indicate the tasks that should be executed before the execution of the task of the current node. Costs can be set to the vertices of the graphs to define the task execution time. It should be mentioned that the costs have been estimated assuming that each task is independently executed from each other. An example of this connected graph is illustrated in Figure 13.

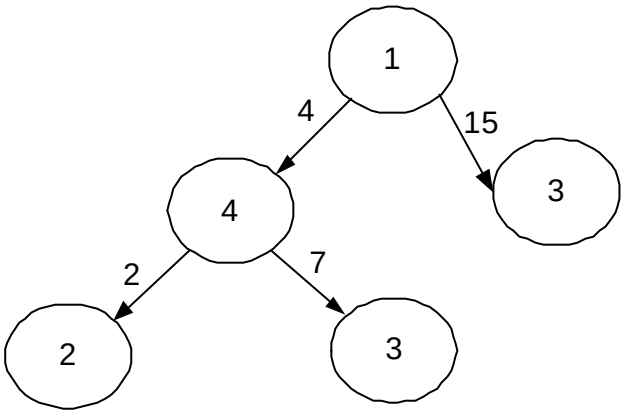


Figure 13: A graph representing tasks dependencies.

To address this case, the execution time of a task taking into account its dependencies is estimated. This is performed by accumulating the cost of all dependent tasks. Then, the critical path is estimated, i.e., the path with the maximum cost. This expresses that the earliest ready time of this task. Therefore, a low bound estimate of the ready time of the tasks. Initially, the algorithm starts to schedule the independent tasks. Then, some dependent tasks become independent, which are next served by updating their respective ready times estimating by the critical path.

References

[1] Foster, I., and Kesselman, C. (editors), *The Grid:Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[2] Leinberger W., Kumar V., Information Power Grid: The new frontier in parallel computing? *IEEE Concurrency*, October-December 1999, 75-84

[3] L. Jackson, G. Rouskas, "Deterministic Preemptive Scheduling of Real Time Tasks," *IEEE Computer* Magazine, Vol. 35, No. 5, pp. 72-79, May 2002.

[4] M. S. Fineberg and O. Serlin, "Multiprogramming for Hybrid Computation," *Proc. of IFIPS Fall Joint Computer Conference*, Thompson, Washington DC, 1967.

[5] J. A. Stankovic et. al., "Implications of Classical Scheduling Results for Real Time Systems," *Computer* pp. 16-25, June 1995.

[6] M. L. Dertouzos and A.K.-L. Mok, "Multiprocessor On-line scheduling f Hard Real Time Tasks," *IEEE Trans. on Software Eng.* pp. 1497-1506, Dec. 1989.

[7] G. Manimaran, C. Siva Ram Murthy, Machiraju Vijay, and K. Ramamritham, "New Algorithms for Resource Reclaiming from Precedence Constrained Tasks in Multiprocessor Real-time Systems," *Journal of Parallel and Distributed Computing*, vol. 44, no. 2, pp. 123-132, Aug. 1997.

[8] K. Ramamritham, J.A.Stankovic, and P.-F. Shiah, "Efficient Scheduling Algorithms for Real-time Multiprocessor Systems," *IEEE Trans. on Parallel and Distributed Systems*, vol.1, no.2, pp.184-194, Apr. 1990.

[9] C. Shen, K. Ramamritham, and J.A. Stankovic, "Resource Reclaiming Inmultiprocessor Real-time Systems," *IEEE Trans. on Parallel and Distributed Systems*, vol.4, no.4, pp.382-397, Apr. 1993.

[10] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer Verlag, 1994.

[11] D. E. Goldberg, Genetic Algorithm in Search, Optimization and Machine Learning, Addison Wesley, 1989.

[12] Dimitri Bertsekas, Robert Gallager, *Data Networks*, 2nd ed., published by Prentice Hall 1992. The
section on max-min fairness starts on p.524.

[13] A.K Parekh, R.G Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Transactions on Networking*, Vol. 1 No. 3, pp. 344 -357, June 1993.

APPENDIX B: Uniform Processor Utilization

Notation

Let us assume a multiprocessor system consisting of $M>1$ processors, denoted as $P_1,\cdots,P_M$ and that each processor its characterized by a capacity C_i, $i=1,\ldots,M$. Our purpose is to schedule tasks T_i, $i=1,2,\ldots,N$ to the M processors or in other words to determine *when and on which* processor a task is executed. In our approach, tasks T_i are aperiodic and are characterized by a deadline, say D_i, and a ready time, say $d_{i.j}$, before of which the respective task T_i cannot be executed on the jth processor. Ready time expresses the earliest time that the task T_i is available for processing on the jth processor, while deadline D_i the time beyond of which the task is prohibited to be executed. Ready time may results from communication delays or other networking constraints, while task deadline usually results from the users' requirements.

Each task demands a number of instructions for its execution and thus a workload w_i if it is executed on a processor of unit capacity. Therefore in case that the task T_i is assigned to the P_j processor of capacity C_j and assuming that it occupies 100% of the processor utilization, the task execution time is

$$w_{i,j} = \frac{w_i}{c_j} \tag{A1}$$

We assume that upon a task arrival, deadline D_i, ready time $d_{i,j}$ and the workload w_i are known to the scheduler. The workload w_i is provided using, for example, a prediction mechanism, which estimates the number of instructions that the task T_i requires for its execution. Considering that the task T_i is executed on the processor P_j with a utilization degree $a_{i,j}<1$, then the computational time of task T_i will be increased by the amount $a_{i,j}$

$$w^a_{i,j} = \frac{w_i}{a_{i,j}\cdot c_j} = \frac{w_{i,j}}{a_{i,j}} \tag{A2}$$

where $w^a_{i,j}$ refers to the execution time of task T_i on processor P_j with utilization $a_{i,j}$.

Figure A1 illustrates how the task execution time is affected by the utilization degree $a_{i,j}$ of task T_i on processor P_j. Since the task execution time $w_{i,j}$ should be smaller than the respective deadline D_i, the utilization used for T_i should be greater than or equal a minimum value (see Figure A1), is defined as

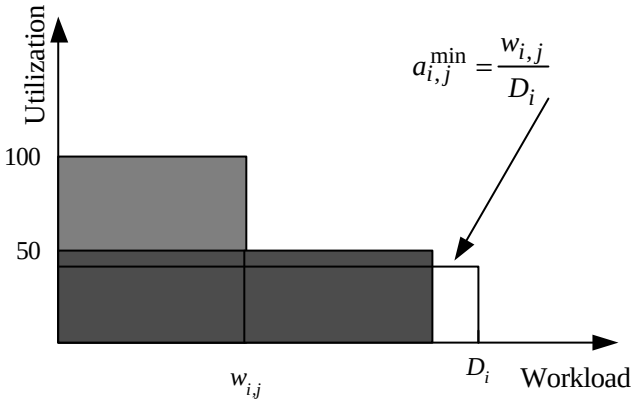


Figure A1. Variation of the task execution time with respect to the utilization degree.

$$a_{i,j}^{\min} = \frac{w_{i,j}}{D_i} \quad \text{(A3)}$$

In case that the workload w_i cannot be predicted, we assume that the statistics of task T_i are known. Particularly, supposing that the average workload $\overline{w}_i$ and the respective deviation s_i are available, we can estimate the worst execution time of task T_i as $w_i^s = \overline{w}_i + g s_i^2$, where factor g determines the confidence we have about the execution time of task T_i.

The following assumptions are also made:

- The tasks are Tasks are non-preemptable and non-interruptible, i.e., once a task starts execution, it finishes to its completion on the same processor and without being interrupting.
- The utilization degree $a_{i,j}$ remains constant during the task execution.

Estimation of Utilization Degree

Let us define as $V = \{T_1, \ldots T_N\}$ a set consisting of all N tasks to be scheduled. Let us assume that k tasks have been scheduled including in the set S_k, while the remaining N-k unscheduled

tasks comprise the set U_k. It is clear that $V = S_k \cup U_k$. Initially no tasks have been scheduled, meaning that $S_{k=0} = \varnothing$ and $U_{k=0} \equiv V$.

A task is said to be feasible if its timing constraint, i.e., the deadline, are met by the scheduler. To examine whether a task is feasible or not, we need to estimate the time that the task starts its execution $r_{i,j}$ and the utilization degree assigned. Then, the task T_i finishes its execution at time $r_{i,j} + w_{i,j}^a$ assuming that T_i is executed on the jth processor P_j with constant utilization a. A task can be feasibly served, if we can estimate a starting time $r_{i,j} \geq d_{i,j}$ and a constant utilization a so as $r_{i,j} + w_{i,j}^a \leq D_i$. Otherwise, it is considered non-feasible. The set S_k is said to be feasible if all the included tasks are feasible.

To estimate the task starting time and utilization a, we need the processor profile. Let us denote as $g_j(t)$ the utilization profile of P_j processor. The $g_j(t)$ is formed by tasks that have been already scheduled on the jth processor. Since each task is served with a constant utilization degree, function $g_j(t)$ presents a step-wise form. In the following, we omit subscript j for simplicity, since we refer to a particular processor. Let us assume that $g(t)$ has zero value for $t>t_N$. This mans that beyond time t_N no tasks have been scheduled on this processor. Then, if $0 = t_0 < t_1 < \cdots < t_N$ are the points at which $g(t)$ changes value, due to either the completion or the starting of a task execution, function $g(t)$ is defined as follows

$$g(t) = \begin{cases} b_i & \text{if } t_i \leq t < t_{i+1} \\ 0 & \text{if } t \geq t_N \end{cases} \qquad \text{(A4)}$$

Let us assume that a new task, say T_m, is to be scheduled. Then, the problem is to find an appropriate time interval that T_l can be feasibly scheduled. Since the task T_m requires at least $a_m^{\min}$ utilization to be executed so that its time constraints are met [see equation (A3)], time intervals with utilization $b_i + a_m^{\min} > 1$ cannot satisfy the T_m requirements and thus should be excluded for searching. Similarly, time intervals lower than the task ready time d_m (the subscript j has been omitted since we refer on a particular processor) and greater than the deadline D_m should be excluded. On the hand, all the remaining intervals are possible intervals for scheduling

the task T_m. This does not mean that the task T_l can be feasibly executed at each possible interval.

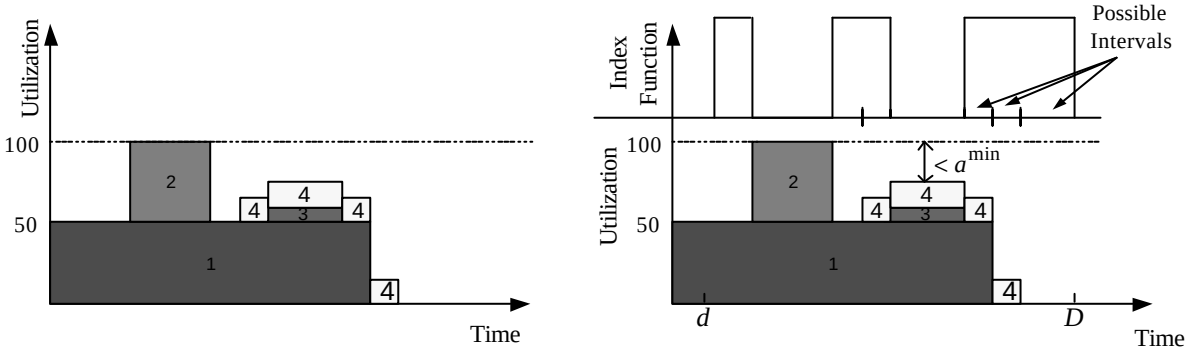
Let us construct in following, an indicator function $I_m(t)$, which takes zero values at time intervals where the new task T_m cannot be scheduled and unit values at time intervals where the T_m is possibly scheduled.

$$I_m(t) = \begin{cases} 0 & \text{if} \quad t < d_m \\ J_k^{(m)} & \text{if} \quad d_m \le t < t_{k+1} \\ J_{k+1}^{(m)} & \text{if} \quad t_{k+1} \le t < t_{k+2} \\ \vdots & \\ J_n^{(m)} & \text{if} \quad t_n \le t < D_m \\ 0 & \text{if} \quad t \ge D_m \end{cases} \tag{A5}$$

and

$$J_i^{(m)} = \begin{cases} 1 \text{ if } b_i + a_m^{\min} \le 1 \\ 0 \text{ if } b_i + a_m^{\min} > 1 \end{cases}, \; i=k,\ldots,n \tag{A6}$$

where we assume that $t_k \le d_m < t_{k+1}$, $t_n \le D_m < t_{n+1}$. Figure A2(b) shows the indicator function for the utilization profile of Figure A2(a).



(a) (b)

Figure A2. (a) A Processor Utilization Profile. (b) The respective Index function.

Time Interval Estimation

Having formed the indicator function for the task T_m, the next step of the algorithm is to detect the time intervals on which the indicator function is equal to one. In particular, let us denote as L_i, i=1,2,…,Q. Intervals L_i are not overlapping meaning that

$$L_i \cap L_j = \varnothing, \ \forall\, i \neq j \tag{A7}$$

Time intervals L_i are defined as the maximum length interval of $g(t)$ on which $I_m(t)$ equals one.

$$L_i = [t_p \ t_q] = [t_p \ t_{p+1}] \cup [t_{p+1} \ t_{p+2}] \cup \cdots \cup [t_{q-1} \ t_q] \quad \text{such that} \tag{A8}$$

$$J_{p-1}^{(m)} = 0, \ J_q^{(m)} = 0 \ \text{and} \tag{A9}$$

$$J_k^{(m)} = 1, \ k = p, p+1, \ldots, q-1 \tag{A10}$$

In the previous equation, we have assumed that the interval L_i is partitioned at time instances $t_p < t_{p+1} < \ldots < t_q$, which actually correspond to time instances at which the utilization profile $g(t)$ changes.

The L_i are possible intervals onto which a given task can be scheduled. Other possible time intervals can be obtained by decomposing interval L_i into its respective partitions. For example, the interval $[t_p \ t_{p+1}] \cup [t_{p+1} \ t_{p+2}]$ of equation (A8), which is a subset of L_i is also a possible interval for scheduling the task T_m.

Let us denote as $Dec(L_i)$ an operator, which returns all possible successive decomposed intervals of L_i. It should be mentioned that the operator $Dec(L_i)$ also returns the set L_i. Then, we can prove the following theorem,

Theorem 1: Let an interval $L_i = L_i^{(1)} \cup L_i^{(2)} \cup \cdots \cup L_i^{(r)}$. Then, operator $Dec(L_i)$ returns $r(r\text{-}1)/2$ possible intervals. This means that the cardinality of $Dec(L_i)$, i.e., $|Dec(L_i)| = r(r\text{-}1)/2$.

Let us denote as L the set containing all possible intervals that the task T_m can be scheduled. Then, $L = \bigcup_{i=1}^{Q} Dec(L_i)$. An interval $l \in L$ is characterized by the respective length, say $len(l)$, and the interval capacity, say $C(l)$. The interval capacity $C(l)$ is defined as the maximum value of the utilization profile over this time interval

$$C(l) = \max g(t)\,,\ t \in l \tag{A11}$$

The utilization degree $a(l)$ which the task T_m requires for its execution in the interval $l \in L$ is given by the following relation

$$a(l) = \frac{w_i}{len(l)} \tag{A12}$$

In order to retain the utilization profile of the processor as low as possible, we find that interval $l \in L$, which minimizes the following equation

$$\hat{l} = \arg\min_{l \in L}\{C(l) + a(l)\} \tag{A13}$$

In case that $C(\hat{l}) + a(\hat{l}) < 1$, the task T_m can be feasibly scheduled to the interval $\hat{l}$. Otherwise, the task T_m cannot be feasibly scheduled at this processor given the utilization profile.

Processor Estimation

In the previous analysis, we have concentrated on a given processor P_j. For this reason, we have omitted the dependence of the variables on the processor index. The most appropriate processor for scheduling the new task T_l is performed similarly, by minimizing the following equation

$$\hat{j} = \arg\min_{j \in P}\{C(\hat{l}_j) + a(\hat{l}_j)\} \tag{A14}$$

where we recall that P is the set of all available processors. The $\hat{l}_j$ indicates that the interval has been estimated by minimizing equation (A14) using the utilization profile $g_j(t)$ of the jth processor.

APPENDIX C: Genetic Based Scheduling

In this chapter, a genetic algorithm is used to perform the scheduling. In this case, possible solutions of the scheduling problem are represented as chromosomes whose "genetic material" corresponds to a specific arrangement of the task on the available processors. In order to apply a genetic algorithm, we need to determine the following: The representation scheme used to encode the genes as tasks' arrangements on the processors; the adopted crossover operator, which generates new chromosomes from one or more chromosomes, the initial population used for the initialization of the algorithm; the mutation scheme, which introduces random gene variations that are useful for exploring new search areas and finally the objective cost function, which is optimized by the genetic scheme.

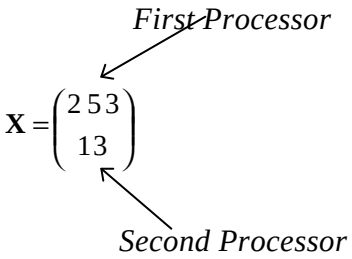


Figure B1. An example of the proposed chromosome representation. Five tasks are assigned to two processors. Particularly, tasks 2, 5, 3 are assigned to the first processor, while tasks 4 and 3 to the second processor.

Chromosome representation

The chromosome representation scheme describes the way that the N available tasks are arranged onto the M processors. In this framework, if a chromosome is known, it is also known how the N tasks are scheduled. In our case, a matrix $\mathbf{X}$ is used for representing the chromosomes. The elements of $\mathbf{X}$, say $x_{i,j}$, take value from 1 to N, i.e., $x_{i,j} \in \{1,....,N\}$ each of which

corresponds to a given task, say T_i, among all N available. Therefore, it is held that $x_{i,j} \neq x_{i',j'}$, $\forall\, i, j \neq i', j'$. The index j of $x_{i,j}$ (columns of matrix $\mathbf{X}$) indicates the processors at which $x_{i,j}$ is assigned to be scheduled, while index i corresponds to the order that the task is to be scheduled. For example, $x_{2,4}$ represents a task that is scheduled second at the fourth processor. Figure B1 presents an example of the proposed chromosome representation in case of processors M=2 and N=5 tasks. In this particular example, tasks 2, 5, and 3 are assigned to first processor, while the tasks 4 and 1 to the second processor.

Initial Population Selection

The genetic algorithm is initialized based on an Earliest Deadline First (EDF) scheme. This scheme also referred as the relative urgency algorithm or the deadline driven rule, dictates that at any point, the system must assign highest priority to the active task with the most imminent deadline. In this way, we can estimate which task of the task queue is to be served (scheduled) first. The next step is to determine at which processor the task should be assigned. If we assume that each task occupies full processor utilization, the processor which provides the earliest starting time of the task execution (EST) is selected as the most appropriate. On the other hand, in case that a utilization degree smaller than 100% utilization is used, the processor is selected using the approach described in the appendix A. Based on the aforementioned statements, an initially chromosome is generated, denoted as $\mathbf{X}(0)$. Other scheduling schemes, such that the Least laxity First algorithm can be used for the initialization of the population.

The initial population affects the number of iterations required for the genetic scheme to converge to the optimal solution. An initial population, which is close to the optimal scheme in general demands much less iterations of the genetic algorithm than a randomly selected initial chromosomes. This is due to the fact that it is more preferable to start from solution that it is expected to be close to the optimal one than from a random one.

Crossover operator

The crossover operator indicates the way that the genes of a chromosome are exchanged in order to produce a new chromosome. A common and simple way to implement a crossover operator is to randomly exchange the genetic material of the chromosome so that new possible search paths are generated. However, faster convergence is achieved if the new chromosomes are

generated so that they present higher probability of being survive (i.e., provide a better solution) than the current examined chromosome.

Let us assume that a given iteration of the algorithm, say e.g., the nth, some tasks have been feasibly scheduled whereas some other have been unscheduled. Let as denote as U the set of all unscheduled task over all processors at the nth iteration, i.e., the set of all non-feasible tasks. Let us also denote as S the set including all feasible tasks over all processors. The goal of the crossover mechanism is to re-arrange tasks so that a better (more feasible) solution is obtained. For this reason, it is more preferable to re-arrange an unscheduled task to a more appropriate position rather than a scheduled one.

Let us assume that without loss of generality the T_m unscheduled task has been selected for re-arrangement. Two different scenarios can be considered. The first assumes that the tasks serve with a full utilization degree of the processor, while the second scenario a different than 100% utilization is assigned for each task.

Full Utilization

Let us recall that the ready time of the selected unscheduled task T_m is denoted as $d_{m,j}$ on the jth processor, while the task execution should be finished earlier than the deadline D_m. As a result, the maximum permitted time interval for being executed the task T_m is within $[d_{m.,j}\ D_m]$. Therefore, it is expected that only the feasible tasks whose the execution time located in $[d_{m.,j}\ D_m]$ affect the T_m execution.

Let us denote as $T_s \in S$ a feasible task whose the starting time $r_{s,j}$ on the jth processor $d_{m,j} \le r_{s,j} \le D_m$ or $d_{m,j} \le r_{s,j} + w_{s,j} \le D_m$. Then, task execution time lies in the interval $[d_{m.,j}\ D_m]$. Let us also denote as $S_{[d_{m,j}\ D_m]}$ the set containing all possible feasible tasks with execution times in $[d_{m.,j}\ D_m]$, i.e.,

$$S_{[d_{m,j}\ D_m]} = \{T_s \in S : d_{m,j} \le r_{s,j} \le D_m \,|\, d_{m,j} \le r_{s,j} + w_{s,j} \le D_m\} \qquad \text{(B1)}$$

where | corresponds to the OR operator.

Then, if $T \in S_{[d_{m,j}\ D_m]}$ is an element of $S_{[d_{m,j}\ D_m]}$ (i.e., a task whose execution time starting or completing in the interval $[d_{m.,j}\ D_m]$), we define the task feasibility according to the laxity

time of the selected task T to the unscheduled task deadline T_m. In particular, the laxity time is defined as

$$d_{T,T_m} = D_m - r_T - w_{T,j} \quad \text{(B2)}$$

where r_T is the starting time of the selected feasible task T, while $w_{T,j}$ corresponds to the workload of the task T as is served by the jth processor. In case, that $S_{[d_{m,j}\, D_m]} = \varnothing$, i.e., no tasks are executed in the interval $[d_{m.,j}\ D_m]$, the laxity time $d_{T,T_m} = D_m$.

All tasks $T \in S_{[d_{m,j}\, D_m]}$ are considered possible tasks for re-arrangement with the non-feasible task T_m. According to the laxity value, a selection probability is assigned. The lower the value of d_{T,T_m} is the more probable is the respective task to be selected. This means that we move the unscheduled task T_m to the least feasible position with a probability value.

For a given feasibly scheduled task, the unscheduled task T_m is positioned to be served at the selected processor *right after* the feasibly selected task. In order to take into consideration the initial positions as well, we can extend the set $S_{[d_{m,j}\, D_m]}$ by "dummy" tasks, which are to be served *before* the first task of a particular processor. As a result, this task is assumed to be located at the 0th column of the arrangement matrix **X**. The laxity time for this "dummy" task is defined as D_m as if this task is executed outside the interval $[d_{m.j}\ D_m]$

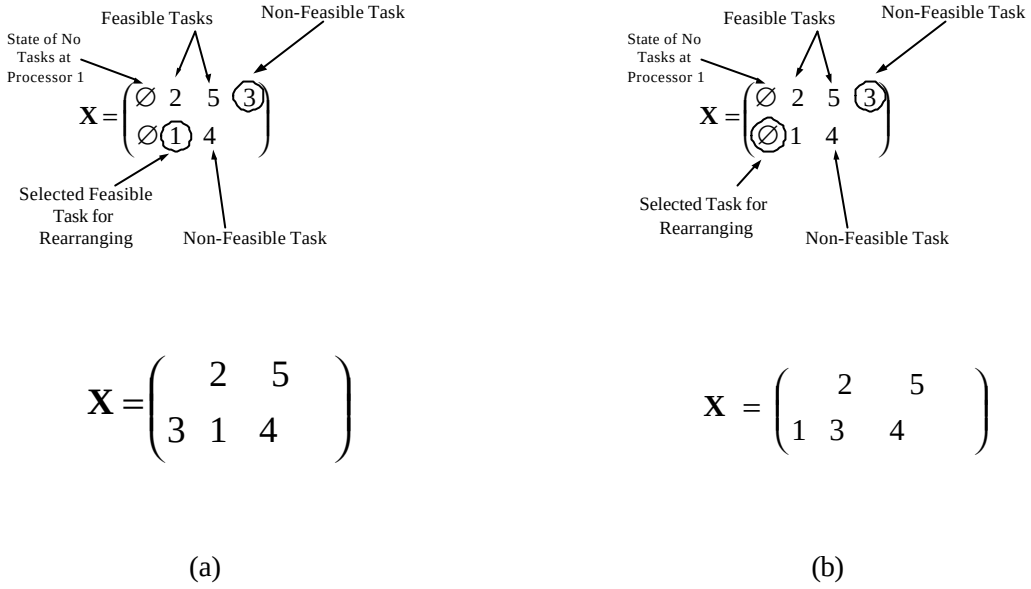
Let us then index all laxity values of tasks $T \in S_{[d_{m,j}\, D_m]}$ as d_1, d_2, …. In this case, for simplicity we have omitted the subscripts that refer to the dependence on the selected feasible task and the examined non-feasible one. Then, a probability can be defined as

$$P_i = 1 - \frac{d_i}{\sum_i d_i} \quad \text{(B3)}$$

The cumulative probability $q_i = \sum_{j=1}^{i} p_j$ is then calculated and a random number $x \in [0,1]$ is generated to select the index at which the unscheduled task is exchanged. The non-feasibly scheduled task T_m is randomly selected from the set U of the unscheduled task at the given chromosome arrangement.

An example of the proposed gene exchange is in Figure B2. Tasks illustrated as $\varnothing$ corresponds to a dummy task so as to examine the case where no tasks are in the interval

$[d_{m,j}\ D_m]$. In this case, we assume that all tasks are in this interval. In the first scenario, the non-feasible task 3 is moved in the right left position of the feasibly scheduled task 1 so that a new arrangement is generated. The task re-arrangement is presented in Figure B2. In the second scenario, the selected task is positioning in the first order of the two processor, since the dummy task is selected.



$$\mathbf{X} = \begin{pmatrix} & 2 & 5 \\ 3 & 1 & 4 \end{pmatrix} \qquad \mathbf{X} = \begin{pmatrix} & 2 & 5 \\ 1 & 3 & 4 \end{pmatrix}$$

(a) (b)

Figure B2. An example of the proposed crossover operator. a) Feasible tasks are possible for re-arranging the selected unscheduled task (3). Task (1) is selected for re-arranging. b) The new chromosome obtained after task re-arrangement.

No Full utilization

A similar crossover operator is adopted even in case that a different degree of utilization can be assigned to each task. As in the previous case, a set containing all feasibly scheduled tasks and a set of all unscheduled tasks are defined, say *S* and *U* respectively. The objective is to re-arrange a non-feasibly scheduled task $T_m \in U$ to a new position so that, after the task arrangement, a better solution is obtained. As is the previous case, it is expected that only the feasible tasks whose execution time starts or finishes within the interval $[d_{m,j}\ D_m]$ defined by the unscheduled task T_m affect its position. For this reason, these tasks are examined, belonging to the set

$S_{[d_{m,j}\, D_m]}$. In this case, however, a different function is adopted to determine the probabilities assigned for each feasible task, which should take into account the utilization degree used.

As we have described in appendix A, each feasibly task is positioned at a processor by exploiting the lowest possible utilization degree. Let us index all utilization degree used for all tasks $T \in S_{[d_{m,j}\, D_m]}$ as a_1, a_2, …. Again, for simplicity we have omitted the subscripts that refer to the dependence on the selected feasible task and the examined non-feasible one. Then, a probability can be defined as

$$P_i = 1 - \frac{a_i}{\sum_i a_i} \qquad \text{(B4)}$$

The previous equation indicates that the feasibly scheduled tasks with lowest utilization degree are more probable to be selected, instead of tasks with the highest degree. The most appropriate utilization degree for the new task position is determined using equation (B4).

As in the previous case, the unscheduled task T_m is positioned to right after the selected feasible task and on the same processor. The initial positions are also included as possible, by extending the set $[d_{m,j}\, D_m]$ by "dummy" tasks as in the previous case.

Another possible position for the task T_m is to be served as the same order and processor, but using a different utilization degree of the tasks ahead it. This is performed in our case according to a probability value say p_u. In particular, if a random number $x \in [0,1]$ is greater than $x > p_u$, the position of task T_m remains the same while the utilization degree of a randomly selected feasible task served at the T_m processor is chosen to change. As utilization, the right higher value is used. Instead if $x \leq p_u$ the previous procedure is adopted.

Fitness Functions and Mutation

The next step of the genetic algorithm is to apply *mutation* to the newly created chromosomes, introducing random gene variations that are useful for restoring lost genetic material, or for producing new material that corresponds to new search areas. *Uniform mutation* is the most common mutation operators and is selected for our optimization problem. In particular, each offspring gene $x_{i,j}$ is replaced by a randomly generated one $x'_{i',j'}$, with a probability p_m. That

is, a random number $x \in [0,1]$ is generated for each gene and replacement takes place if $r > p_m$; otherwise the gene remains intact.

Once new chromosomes have been generated for a given population $\mathbf{X}(n)$, $n \geq 0$, the next generation population, $\mathbf{X}(n+1)$, is formed by inserting these new chromosomes into $\mathbf{X}(n)$ and deleting an appropriate number of older chromosomes, so that each population consists of a constant number of members, in our case equal to the number of tasks *N*. Several GA cycles take place by repeating the procedures of fitness evaluation, parent selection, crossover and mutation, until the population converges to an optimal solution. The GA terminates when the best chromosome fitness remains constant for a large number of generations, indicating that further optimization is unlikely.

At each stage of the algorithm, the task arrangement obtained by matrix $\mathbf{X}$ is evaluated using a cost function and the best solution over all iterations is retained as the most appropriate.

APPENDIX D Fair Scheduling

Notation

In this section, we face the scheduling problem from a different point of view by handling it as an admission control problem. In particular, the aforementioned described approaches a) select an appropriate task for scheduling among all the unscheduled ones and b) estimate an appropriate time interval for executing this task on a suitable resource so that the respective time constraints, i.e., the ready time and deadlines, of the given task are met. Instead, in this alternative approach, the scheduling is performed based on the demanded task rates, which is defined as the fraction of the task workload over the time that the task is permitted to be executed.

$$X_i = \frac{w_i}{D_i - d_i} \quad \text{(C1)}$$

In equation (C1), d_i is defined as the weighted average of ready times $d_{i,j}$ of the ith task over all available processors, so that the demanded task rate is independent from the processor that the task is assigned to.

$$d_i = \frac{\sum_j d_{i,j} C_j}{\sum_j C_j} \quad \text{(C2)}$$

where C_j refers to the processor capacity.

Problem Formulation

The X_i expresses the rate that the processor should allocate for executing the demanded task T_i and is independent of the way that the task T_i is processed. For example, the task can be executed with different utilization degrees but the total rate is equal to the demanded one.

Let us also assume that P processors are available and each processor is characterized by a capacity, say C_i, expressing the number of instructions that the processor can execute.

Our goal is to assign the tasks T_i, i=1,2,…,N to the P available processors so that the tasks are properly executed. However, this requirement cannot be always satisfied. For example, in case that the sum of demanded rates are greater than the total capacity offered by the processors, i.e.,

$$\sum_{i=1}^{N} X_i > \sum_{j=1}^{P} C_j \qquad \text{(C3)}$$

then some (or all) tasks will be executed with a rate less than the demanded one. The execution time of tasks, which are served with a lower rate than the demanded one, increases implying that the respective time constraints are violated. Furthermore, even in case that the overall processor capacity is greater than the total demanded rate, i.e.,

$$\sum_{i=1}^{N} X_i \le \sum_{j=1}^{P} C_j \qquad \text{(C4)}$$

it is probable for some tasks to be executed with a rate lower than the required one. This is due to the fact that in our case we assume that a task is entirely served by one processor. This assumption is reasonable since once a task is assigned to a processor, this processor should complete the task execution.

In case that the demanded rates cannot be fulfilled, reduction of the task rates is accomplished so that the processor capacity constraints are satisfied. In our case, the tasks are modified using a fair policy obtained by the Max-Min fair sharing algorithm. Let us denote as $\hat{r}_i$ the fair task rates. These rates are derived by fairly sharing the processor capacity to the available task according to the contribution of the users to the Grid resources. Tasks, whose the demanded rates X_i are less than the initial capacity rate sharing to this task, are served with the demanded rates. The remaining capacity is sharing to unsatisfied tasks according to their contribution to the total resources.

In case that equation (C4) is satisfied, the Max-Min fair sharing algorithm leaves the demanded rates unchanged. This means that in this case the fair task rates $\hat{r}_i = X_i$ for all tasks N. Otherwise, the task rates X_i are modified resulting in the fair task rates $\hat{r}_i$, which satisfy the following relationship

$$\sum_{i=1}^{N} \hat{r}_i = \sum_{i=1}^{P} C_i \quad \text{if} \quad \sum_{i=1}^{N} X_i > \sum_{j=1}^{P} C_j \qquad \text{(C5)}$$

The Max-Min fair sharing algorithm can be successfully applied for task scheduling in case that the rate of a task can be split in any piece and then process by any one of the available processors. In our case, however, we assume that the rate of one task can be only served by one processor. As a result, it is probable the scheduled task rates, say r_i, are different than the rates $\hat{r}_i$, even in case that equation (C4) is satisfied.

Therefore, our goal for a fair scheduling policy is to estimate the scheduled task rates r_i so that they are as close as possible to the fair task rates $\hat{r}_i$ and simultaneously the processor capacity constraints are not violated under the condition that each task is scheduled only to one processor. These conditions are expressed in the following equations

$$\min \left\| \mathbf{r} - \hat{\mathbf{r}} \right\|_p \tag{C6a}$$

$$\sum_{i \in B_j} r_i \le C_j , \qquad B_j = \{ i : T_i \text{ scheduled on } j \text{ processor} \} \tag{C6b}$$

$$r_i \le \hat{r}_i , \text{ for all } i \tag{C6c}$$

where $\mathbf{r} = [r_1, \cdots r_N]^T$ is a vector contains the scheduled rates for all N tasks, while $\hat{\mathbf{r}} = [\hat{r}_1, \cdots \hat{r}_N]^T$ a vector of the respective task fair rates. The $\left\| \right\|_p$ indicates the p-norm and the set B_j contains the indices of tasks that have been scheduled to the jth processor. Equation (C6b) means that, after scheduling, the task rates should satisfy the processor capacity constraints. The constraint imposed by equation (C6c) indicates that the scheduled rates should be smaller or equal to the fair rates. Initially, the scheduler assigns the tasks using as rates the fair ones $\hat{r}_i$ so that equation (C6a) is minimized. In case that equation (C6b) is not valid, the scheduled rates should be reduced to satisfy the constraint (C6b). Otherwise, there is a remaining capacity on the jth processor and, since the error of (C6a) is zero over this processor, there is no reason to increase task rates beyond the fair rate $\hat{r}_i$. Instead, the remaining capacity can be used for reducing the errors over other processors (stem from processor overflow) by re-arranging, for example, the tasks.

In case that the p=1-norm is used for the minimization, equation (C6a) is written as

$$\min \sum_{i=1}^{N} (\hat{r}_i - r_i) = \min \sum_{i=1}^{N} e_i \tag{C7}$$

In equation (C7), the absolute operator has been ignored since $r_{i\le}\hat{r}_i$ for all i as equation (C6c) indicates. However, minimization of equation (C7) *does not yield a fair scheduling policy*. This is due to the fact that (C7) expresses the total error over all task rates, and therefore it is possible some tasks are handled in a more favor way against other tasks. For example, let us assume that two tasks are to be scheduled and the minimum p=1-norm is equal to one (1). Then, errors $e_1 = 1$ and $e_2 = 0$ yield total error equal to one but unfairly handle the two task rates. Instead, a more fair solution can be obtained by setting the errors $e_1 = 0.5$ and $e_2 = 0.5$.

A fair error sharing is provided through the $p = \infty$-norm

$$\min \max_i (\hat{r}_i - r_i) = \min \max_i e_i \qquad \text{(C8)}$$

Equation (C8) indicates that the optimal solution for the scheduled task rates is not the one, which minimizes the sum of errors over all available tasks but the one, which yields the minimum reduction of the maximum error of the task errors.

Task Re-arrangement

Minimization of equation (C6) is a computationally intensive problem, which resembles to the bin-packing algorithm. This means that the optimal solution can be found if all possible combinations of the task arrangement should be examined. However, this is practically impossible especially for large number of tasks and processors. In the following, we propose a task re-arrangement scheme of polynomial complexity, so that the total error expressed either (C7) or (C8) is reduced.

The idea behind the proposed task re-arrangement scheme is to efficiently combine processors overflows with processor underflow so that a better exploitation of the overall processor capacity is accomplished. In particular, let us denote as O_j and U_k a processor overflow and underflow respectively,

$$O_j = \sum_{i \in B_j} \hat{r}_i - C_j, \quad j : O_j > 0 \qquad \text{(C9a)}$$

$$U_k = \sum_{i \in B_k} \hat{r}_i - C_k, \quad k : U_k < 0 \qquad \text{(C9b)}$$

Then the following theorem is valid.

Theorem 1: If there is an overflowed processor j, then there always exist at least another underflowed processor $k, k \neq j$.

Using this theorem, we cam re-arrange the tasks in order the overall capacity overflow. This can be performed by compensating processor capacity overflow with processor capacity underflow. In particular, let us also assume that we substitute a task rate r_l of the overflowed processor with a rate r_k of the undeflowed processor. It is clear that, after the task re-arrangement, the processor overflow and underflow are updating as follows

$$O'_j = O_j - c \tag{C10a}$$

$$U'_k = U_k - c \tag{C10b}$$

where $c = \hat{r}_m - \hat{r}_l$ expresses the task rate difference and O'_j and U'_k the updating processor residual. Rates $\hat{r}_m$ and $\hat{r}_l$ can take zero values to include the case that a task can be moved from a processor to another processor. For example, supposing that $\hat{r}_l = 0$, we have the case that the task with fair rate $\hat{r}_m$ leaves the jth processor to be assigned to the kth processor, without substituting any task to processor k.

Taking into consideration equation (C10), we can estimate appropriate bounds for the task rate difference c so that, after the re-arrangement, reduction either of (C7) or (C8) is accomplished.

Theorem 2: A task substitution with $0 < c < |U|_k + O_j$, leads to a reduction of the $|\mathbf{r} - \hat{\mathbf{r}}|_1$ and thus to a better solution.

Theorem 3: A task substitution with $c \in \left[\left(\frac{n'_{\hat{j}}}{n_{\hat{j}}} + 1 \right) O_{\hat{j}} \quad |U|_k + \frac{n'_k}{n_{\hat{j}}} O_{\hat{j}} \right]$ leads to a reduction $|\mathbf{r} - \hat{\mathbf{r}}|_\infty$. The n_j and $n'_{\hat{j}}$ are the number of tasks assigned to the $\hat{j}$-th processor before and after the task re-arrangement, while n_k and n'_k the respective number of tasks on the k-th processor. The $\hat{j}$-th processor is the processors of maximum overflow before the task arrangement, while k is an underflowed processor.

Fair Error Sharing

In case that the tasks rates scheduled on a processor are greater than the processor capacity, reduction of the task rates should be performed. The way that the task rates are reduced depends on the adopted scheduling policy.

The function used for describing how close are the scheduled rates to the fair rates (the adopted norm) defines the adopted scheduling policy. In particular, let us assume that the scheduled rates are estimated so that the overall processor overflow (error) is minimized. This is expressed by equation (C7). Then, the overflow can be randomly distributed to over the respective processor tasks. For example, in this approach it is equivalent to assign the overall error to one task (if this is possible) or to equal sharing the error over all tasks of the respective processor. It is clear that such an approach does not yield a fair scheduling policy.

On the other hand, using equation (C8) a fair scheduling policy can be obtained. This is due to the fact that, in this case, the error for each individual task should be as low as possible. Particularly, let us denote as g_i the weights of the tasks, which express the user contribution, which submits the tasks, to the Grid resources. It is anticipated that tasks submitted by users, which contribute more to the task resources, should favor against the remaining tasks.

For a given task arrangement, the errors e_i are also satisfied the following constraint

$$\sum_{\substack{i \text{ in an overflowed} \\ \text{processor}}} g_i \cdot e_i = O_j \qquad \text{(C11)}$$

The minimum $p = \infty$ norm of errors e_i over an overflowed processor j, is given by the following constraint minimization

$$\hat{e}_j = \underset{\substack{i \text{ in an overflowed} \\ \text{processor}}}{\arg\min \max} \{e_i\} \qquad \text{(C12)}$$

subject to

$$\sum_{\substack{i \text{ in an overflowed} \\ \text{processor}}} g_i \cdot e_i = O_j \qquad \text{(C13)}$$

is provided by weighted sharing the overflowed quantity O_j to all tasks scheduled on this processor. This results to the following error sharing

$$\hat{e}_{i,j} = \frac{O_j}{n_j \cdot g_i} \qquad \text{(C14)}$$

In equation (C14), n_j expresses the number of tasks on the jth processor. In case that all users are equally contribute to the system resources (i.e., g_i=1), the processor overflow is equally sharing to all tasks scheduled on this processor.