



DELIVERABLE D3.3
PROTOTYPE DOCUMENTATION
ON SCHEDULING AGENTS

WP3.2

Document Filename:	CG3.2-D3.3-v1.2-UAB010-PrototypeDescription.doc
Work package:	WP3
Partner(s):	UAB, CSIC, Datamat
Lead Partner:	UAB
Config ID:	CG5.2-Tem-v1.2-UAB-PrototypeDoc
Document classification:	PUBLIC

Abstract: This document describes the main components developed in the first prototype of CrossGrid Scheduling Agents. The document also includes references to the documents that describe the installation procedure.

Delivery Slip

	Name	Partner	Date	Signature
From	Task 3.2	UAB, Datamat, CSIC	15.05.2003	Miquel Senar
Verified by	Norbert Meyer	PSNC	21.05.2003 (version 1.2)	Miquel Senar Norbert Meyer
Approved by				

Document Log

Version	Date	Summary of changes	Author
1.0	17-01-03	First Draft	Miquel Senar, Alvaro Fernandez, Marco Sottilaro, Elisa, Heymann, Jordi Bergés
1.1	21-02-03	Added several corrections according to comments made by reviewers from IRB.	Miquel Senar, Alvaro Fernandez, Marco Sottilaro, Elisa, Heymann, Jordi Bergés
1.2	15-05-03	Added corrections according to comments made by EU reviewers. The main comments has been added to Executive Summary and in Section 2.1	Miquel Senar

CONTENTS

1. EXECUTIVE SUMMARY	4
2. INTRODUCTION	5
2.1. PURPOSE	5
2.2. DEFINITIONS, ABBREVIATIONS, ACRONYMS.....	6
3. REFERENCES	7
3.1. SOURCE CODE	7
3.2. CONTACT INFORMATION.....	7
3.3. REFERENCED DOCUMENTS	7
4. IMPLEMENTATION STRUCTURE	8
5. PROTOTYPE FUNCTIONALITY	10
6. USER MANUAL	12
7. INTERNAL TESTS	13
8. ISSUES	14

1. EXECUTIVE SUMMARY

This document describes the main components of the first prototype of CrossGrid Scheduling Agents. Sections 1 and 2 provide an introduction and references. Section 3 describes the implementation of the modules that are currently available. Section 4 describes the functionality. Section 5 contains the user manual information. Section 6 describes the tests that have been carried out to verify the correct behaviour of the prototype.

The current document has been slightly modified according to the review document that mentions that “the scope of Task 3.2 was greatly reduced” because the work reported here does not include any notion of self-adaptive agents. However, we would like to point out that the scope of the task has not been reduced and the work carried out during the first year has been targeted to the same objectives as were stated in the Technical Annex.

In section 2.1, we are providing technical explanations that summarize the goal of task 3.2 according to the Technical Annex description and the actual achievements in the first prototype, which are consisted with the original goals. These explanations and other comments included in Deliverables D3.1 and D3.2 justify some of the design decisions made for the first prototype.

As said in previous documents, it is worth noting that the task has tried to avoid duplication of efforts by repeating parts of the work done in other projects. In particular, Task 3.2 has taken the EDG’s Resource Broker (release 1.x) as a base software element, which has been augmented and enhanced with the functionality required to fulfill specific CrossGrid requirements. We think that this approach should reduce the overall risk of the Task, and it will increase compatibility between CrossGrid and DataGrid testbeds and software tools, although a non negligible effort has been devoted to analyze, modify and troubleshoot EDG software (which is still under a process of continuous improvement and bug-fixing). We also believe that this approach will allow Task 3.2 to concentrate on key scheduling functionalities required by CrossGrid applications that are not provided elsewhere.

Work in the first prototype was focussed on providing basic scheduling services to support parallel applications written in MPI. These scheduling services do not exhibit a full adaptability, in the terms described in the Technical Annex. They are able to schedule applications according to user preferences and information obtained from the Information Index, but this information is mostly static and, therefore, the scheduler exhibits a limited ability to adapt himself to dynamic changes in the Grid. In order to make scheduling services more adaptive, dynamic information obtained by monitoring tools is required. However, such information is not still available because CrossGrid monitoring and performance prediction tools have not been completely deployed in the Grid testbed and there is no common interface to retrieve information produced by them. As more information is provided by the monitoring tools about the availability and expected performance of Grid resources, our scheduling services will be able to fully adapt their decisions to changing conditions in the Grid. According to the time schedule of monitoring tools, we expect that a first prototype of adaptive schedulers will be available at the end of the second year and further developments will be carried out during the last year of the project.

The work during the first year was also devoted to analyze specific requirements made by application groups of WP1 that were not included in the initial description of Task 3.2 in the Technical Annex. Some prospective solutions to these requirements were described in deliverables 3.1 and 3.2, and will be available in future prototypes.

2. INTRODUCTION

2.1. PURPOSE

The first prototype of Scheduling Agents is intended to provide support for submission of MPI jobs over the Grid. It has been built on top of EDG Workload Resource Management software (specifically, version EDG 1.2.2).

According to the Technical Annex, the main objective of Task 3.2 is to “develop a Grid resource management system based on self-adaptive scheduling agents for scheduling a particular parallel application submitted to a Grid, with the goal of achieving a reasonable trade-off between resource usage efficiency and application speedup, in accordance with the user's optimisation preferences”. The *agent* is the element that is basically responsible for remembering jobs in persistent storage while finding resources to run them. The Resource Broker of EDG constitutes an example of *agent* for sequential applications. Like traditional local schedulers (such as Condor, LSF, PBS, etc.) it maintains a permanent queue of jobs. In contrast to local schedulers, it includes additional functionality to find resources over the Grid. And it uses another external agent, Condor-G, which is responsible for initiating, monitoring and managing the execution of the application on Grid resources.

The Resource Broker of EDG has a scheduler that is targeted to sequential jobs that are executed in a batch fashion. The CrossGrid project requires a scheduler targeted to parallel and interactive jobs. According to the functionality provided by EDG's Resource Broker, Task 3.2, on the one hand, has kept a set of common services that may be shared by both kinds of jobs. On the other hand, new services required by parallel and interactive applications that do not have support in existing projects were included.

Basic services inherited from the Resource Broker include:

- the queue management service that maintains jobs in persistent storage.
- a logging and bookkeeping service that maintains a log for all jobs.
- Condor-G as an agent that executes user applications on remote resources.
- a basic interface that includes user commands to submit jobs and query their status.
- an interface with the Information Index of EDG, which guarantees compatibility with EDG's testbed.

New services developed by Task 3.2 that were not available in the Resource Broker include:

- a new selection service to find suitable resources for parallel applications (i.e. MPI). This service implied also the need for some extensions in the Job Description Language used to specify user's applications.
- a new scheduling policy that maps the user application to a specific set of resources obtained by the selection service mentioned above. EDG's Resource Broker currently uses a basic strategy in which jobs are submitted “close” to data. Our new strategy is mainly based on preferences indicated by the user, and the immediate availability and locality of resources.
- integrated interface between Roaming Access Server and Resource Broker. This interface (that, according to the Technical Annex, was originally planned for month 24) has been completed to a significant extent. This change on the time schedule was adopted in order to provide users with a working prototype over the Grid testbed as soon as possible.

2.2. DEFINITIONS, ABBREVIATIONS, ACRONYMS

CE	Computing Element
CG	CrossGrid Project
EDG	DataGrid Project
JDL	Job Description Language
JSS	Job Submission Service
MPI	Message Parsing Interface
PS	Portal Server
RAS	Roaming Access Server
RB	Resource Broker
SA	Scheduling Agent
WN	Working Node

3. REFERENCES

3.1. SOURCE CODE

Source code of the prototype is based on EDG distribution of the Workload module. Based on this distribution, changes have been made to provide the extra functionality so we will call this release as prototype_1.

The source files that have been modified from the original EDG distribution to support selection of multiple CE are:

```
/Workload/Broker/Server/matchmaking.h  
/Workload/Broker/Server/matchmaking.cpp  
/Workload/Broker/Server/rbcommon.h  
/Workload/Broker/Server/RBjob.h  
/Workload/Broker/Server/RBjob.cpp
```

The modifications on the source code support the selection of more than one CE for the execution of MPI parallel jobs, and these modifications are reached when one JDL description that specifies this kind of job is submitted to the RB.

The main part of the code related to the multiple selection and matchmaking can be found in the file ``/Workload/Broker/Server/matchmaking.cpp``.

3.2. CONTACT INFORMATION

Please provide contact information for partners responsible for the prototype and for technical staff.

Multiple CE selection and matchmaking Álvaro Fernández <Alvaro.Fernandez@ific.uv.es>

3.3. REFERENCED DOCUMENTS

CGTA Annex I – Description of Work
 CrossGrid, November, 2001.
CGSRS32 Task 3.2 Grid Resource Management SRS
 CG-3.2-SRS-0010.doc, 2002.
CGSDD32 Task 3.2 Scheduling Agent Design Document
 CG3.2-D3.2-v1.0-UAB020-SchedulingAgentsDesign.doc, 2002.
CGIG32 Task 3.2 Installation Guide
 CG3.2-D3.3-v1.0-CSIC-installationguide.doc

4. IMPLEMENTATION STRUCTURE

In the design document [CGSDD32] it was stated that Scheduling Agents are responsible for optimising scheduling and node allocation decisions. The main structure of the software was the following:

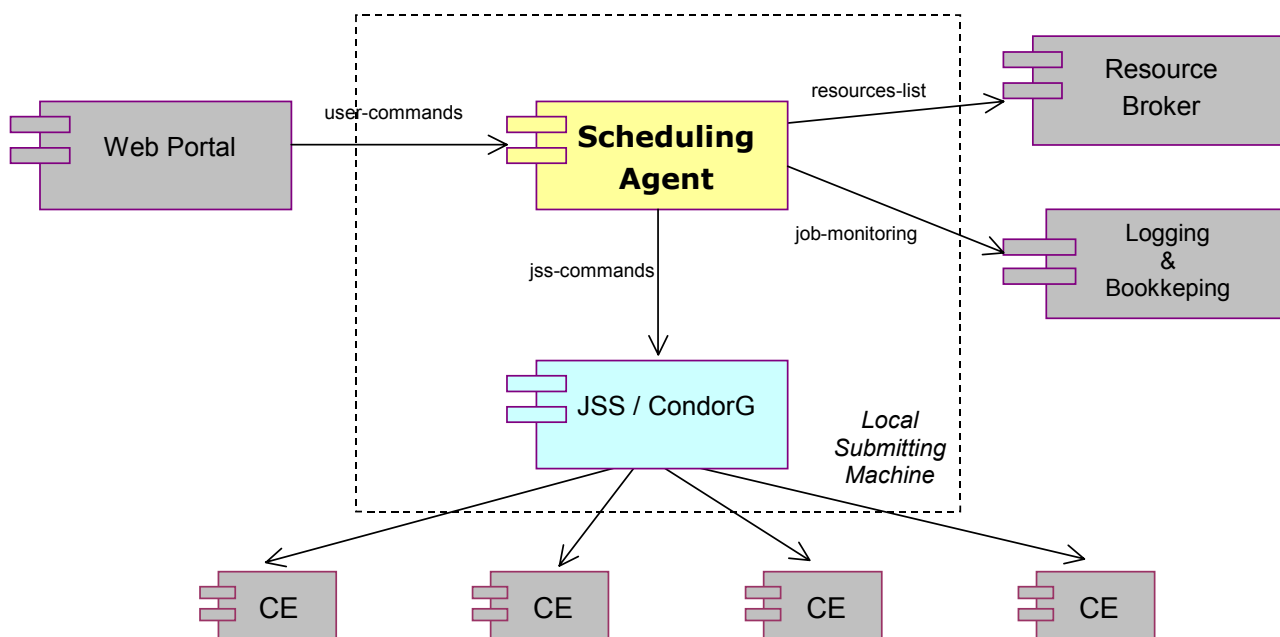


Figure 1: Data flows for Scheduling Agent

As described in previous documents [CGSRS32, CGSDD32], this architecture has many common services with the one currently available in EDG. In our architecture, there is a Scheduling Agent (which is responsible for maintaining a user's job queue and all the scheduling decisions) and a Resource Broker (which is responsible for searching available resources according to the queries carried out by the Scheduling Agent). In EDG, both functionalities are combined into a single element (called Resource Broker).

It is worth noting that the existence of a single entity (EDG Resource Broker) or two entities (Scheduling Agent and Resource Broker) does not imply the existence of any additional functionality in terms of scheduling capability. This is an architectural design difference that might be important mainly from the reliability and scalability point of view [CGSDD32]. According to [CGTA], scheduling parallel applications on Grid resources is one of the main goals of task 3.2. Therefore, in this first prototype, work has been focussed on the design and implementation of scheduling policies and allocation mechanisms to support parallel applications written in MPI (mainly, linked with the MPICH-G2 library). In addition to these MPI support services, work has also focussed on the

implementation of interface services between the user Web Portal/Roaming Access Server and the scheduler services. These mechanisms have been added to the currently available EDG Resource Broker (release 1.2.2, specifically). The fully distributed architecture with separate Scheduling Agents and Resource Broker is planned for the second prototype.

The use and integration of scheduling policies into the workload management services of EDG has simplified the use of several services (such as the Information Index service and the Logging and Bookkeeping service) that are not implemented nor maintained in the CrossGrid project. It has also avoided duplication of efforts for providing some basic services (such as the job queue maintenance) that do not provide any additional functionality for scheduling parallel jobs. And it has allowed the implementation of a nearly complete operational solution for parallel job execution on a grid infrastructure in a very short time period. In practice, the extension of existing EDG services with the new functionality should allow a quick deployment of the first prototype over a distributed grid (instead of a single cluster as it was planned in the Technical Annex [CGTA]). By doing so, the new services included to support parallel jobs over the grid will be better integrated, tested and refined with the use of the CrossGrid testbed (currently based on EDG release 1.2.2 and 1.2.3) and the applications implemented in WP1, which will be available at month 18.

The current architecture is as depicted in figure 2.

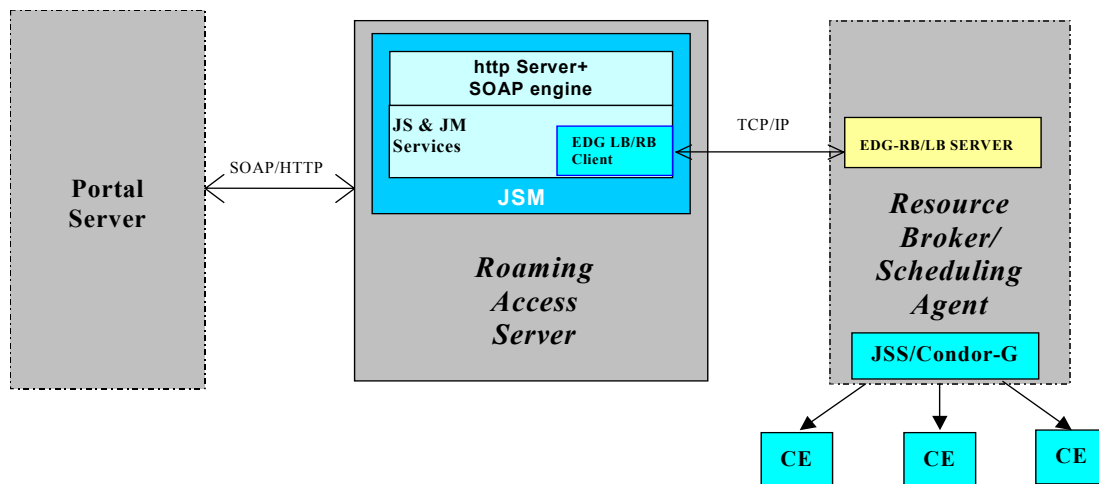


Figure 2: High-level view of Scheduling Agents, PS and RAS components.

The main components included in the first prototype is a matchmaking service that gives support for selecting multiple CEs in the case of MPI jobs.

A second component that is not yet fully operational is an MPI application launcher that is required to start the MPI application over the sites selected by the matchmaking service. This module is based on the grid implementation of MPICH (MPICH-G2). However, it still has several problems due to the limitations of MPICH-G2 (and Globus) to run with private IP addresses, which is the most common scenario in the current CrossGrid testbed.

5. Prototype Functionality

The functionality included in the current prototype is the Resource Selection as defined in [CGSDD32]. This part provides an automated mechanism for selecting groups of CEs that match the requirements of an MPI job, and that will be used as part of the Scheduling Agent that finally submits the job to the selected CEs.

Currently, the Scheduling Agent selects a CE or a group of CEs on which to run the MPI job, according to the following criteria:

1. Groups of CEs with fewer numbers of different CEs are selected first. This criterion tends to run MPI on a single cluster, which will avoid large message latencies between tasks allocated in different clusters.
2. If there is more than one group with the same number of CEs, the group having best global rank is selected first. Ranks are assigned to each CE (or groups of CEs) according to certain performance metrics (e.g. overall MFLOPS, main memory, etc.) specified by the user.

The scheduler passes each CE or group of CEs to the application launcher module in order. If the job is not effectively launched on the selected resources, the scheduler simply passes the next choice according to the previously mentioned criteria. This process is carried out until the job is successfully launched or the list of suitable resources is exhausted. In the latter case, the job is put on a waiting state and it will go through the whole resource selection process again.

The selection of resources is carried out according to the following steps:

- First step: a list is obtained of single CEs that fulfill all job requirements referring only to required individual characteristics (*suitableCEs*). Currently, these are the requirements that are specified in the *Requirements* section of the file describing the job using JDL. This step constitutes a pre-selection phase that generates a reduced set of resources suitable for executing the job request in terms of several characteristics such as processor architecture, OS, available physical memory, etc.
- Second step: from the list mentioned above, groups of CEs are made to fulfill collective requirements. For example, an attempt is made to fulfill the total number of CPUs required by a job by “aggregating” individual CEs. In the case of the number of CPUs required by the job, for instance, the Resource Searcher aggregates CEs to guarantee that the total number of free CPUs in the groups of CEs is larger than *MinNumCPU*, as described in the *JobAd*.

In the second step, there are two main structures used for each submitted job:

- *matchedListCEs*: a vector that comprises the groups of CEs that fulfill all the requirements.
- *partiallyMatchedCEs*: a vector of partial groups of aggregated CEs that still do not fulfill all the requirements, but which are suitable candidates if they are aggregated with other suitable CEs.

The selection process goes through the list of single CEs (*suitableCEs*) starting from the beginning. If the evaluated CE fulfils all the requirements, it is included in the *matchedListCEs* (in the first entry,

which corresponds to groups with only 1 element), and the process continues with the next entry in *suitableCEs*.

If the evaluated CE does not fulfill the requirements by itself, the matching algorithm goes across the groups of CEs included into *partiallyMatchedCEs* to see whether, by adding this CE to each one of the groups in the vector, a group that fulfils the requirements can be formed. If the new group is found, it is inserted in the corresponding entry of *matchedListCEs* according to the number of elements in the group (second entry, for groups of two CEs; third entry, for groups of three CEs; and so on). If the new CE combined with a group included in *partiallyMatchedCE* does not still fulfill the requirements, the CE is added to that particular group and it is checked with the next group in the *partiallyMatchedCE* list.

As described above, our current search procedure is not exhaustive, as it does not compute the power set of all CEs. This means that, for instance, if two solutions are provided like: {CE2}, {CE1, CE3} , the solution that includes {CE1, CE2, CE3} is not considered because one subset of the CEs has already been included in a previous group. In principle, this characteristic should not prevent our Resource Selection module from obtaining good selections. CEs are sorted according to a Rank expression provided by the user in the JobAd. According to the Rank expression the Resource Selection module sorts the suitable CEs in descending order. This means that the most desirable CEs or groups of CEs will be first.

6. User Manual

See document CGIG32.

7. Internal Tests

Functional testing has been conducted in the Resource Selection module both using “black box” techniques and “white box” techniques. In the first case, a job described by its JDL was submitted to the RB and correct behaviour of the Resource Selection was assessed by checking the final list of selected resources. In the second case, the internal operation of the Resource Selection module was tested using a log file.

A practical example for testing the Resource Selection module is included in document CGIG32.

8. ISSUES

The primary issue is that the current prototype is not fully operational although the main component is working in accordance with the design document. The missing component that is still required by the Scheduling Agent is an MPI application launcher. This component is not part of the scheduler itself: it is an external component that carries out all the necessary steps that provide a reliable submission service of the application on the remote resources (including all necessary file staging and error reporting).

Currently, it would be possible to use a simple *globusrun* command as a launcher. However, this method exhibits significant limitations in terms of reliability and error recovery. Condor-G is an alternate launcher that overcomes this limitations by providing fault tolerance and exactly-once execution semantics for grid jobs. It has been designed primarily as a reliable front-end to a computational grid and is currently being used as part of the EDG launcher for sequential jobs. Unfortunately, there is no support in Condor-G for parallel jobs running over multiple sites.

Therefore, work is currently being carried out to provide a reliable application launcher for MPI jobs, based on the services that Condor-G provides for sequential jobs.

It is also worth noting that the MPICH-G2 library, which is commonly used to implement MPI applications to run on the Grid, is currently limited to running on machines that have public IP addresses. This means that the application launcher is going to use two launching mechanisms: the first mechanism is applied to launch MPICH applications (compiled with the P4 device) into single clusters in which internal machines have private IPs. The second mechanism is used to launch MPICH applications compiled with the G2 device over multiple nodes that have public IPs. Due to the existence of these two different launchers, work is also in progress into the Resource Selection module to restrict the search between CEs with public or private IPs, according to the kind of MPI application submitted. The `jobtype` field of the `JobAd` will be modified accordingly. All these changes will be included into the Resource Selection module in the near future.