An Automated Cluster/Grid Task and Data Management System

Luís Miranda, Tiago Sá, António Pina, and Vítor Oliveira

Department of Informatics, University of Minho, Portugal
{lmiranda,tiagosa,pina,vspo}@di.uminho.pt

Abstract. CROSS-Fire is a research project focused on wildfire management related topics, that defines an architecture integrating three main components: the CROSS-Fire Platform, implemented as an OGS-WS compatible Web Processing System layer; a Spatial Data Infrastructure platform; a Distributed Computing Infrastructure, supporting both cluster and grid environments.

The main objective is to exploit Cluster/Grid middleware in order to: i) allow the concurrent execution of multiple and large simulations, ii) access and manage large data input/output files, iii) create and handle a database of past simulations and iv) allow remote and interactive monitoring of the fire simulation growth.

The developed tools manage the tasks and control the data flow between components, providing the advantage of fast implementation and easy testing, on cluster and grid environments.

Key words: Civil Protection, Cluster, Ganga, Grid

1 Introduction

The CROSS-Fire research project[1], focused on Civil Protection topics, particularly wildfire management, defined an architecture that integrates three main components: i) the CROSS-Fire Platform, ii) a Spatial Data Infrastructure (SDI) and iii) a Distributed Computing Infrastructure.

The platform is implemented as an OGC-WS compatible Web Processing Service (WPS) layer, that deals with the functionalities of its three components: Business Logic, Geospatial services and Cluster/Grid services. The Business Logic is configured to handle the wildfire management algorithms of the platform, namely, forest fire propagation and wind field calculation. We describe a fast implementation and easy testing platform that uses either a cluster or the EGEE/EGI and the interaction between the WPS, a data storage medium and an application execution environment.

The CROSS-Fire platform executes fire and wind simulation applications, requiring powerful computational resources and produces a considerable amount of data that needs to be accessible everywhere. The main objective for exploiting Cluster/Grid middleware is: i) to allow the concurrent execution of multiple and

[1] CROSS-Fire project homepage: https://pop.cp.di.uminho.pt/crossfire/

large simulations, ii) to access and manage large data input/output files, iii) to create and handle a database of past simulations and iv) to allow remote and interactive monitoring of the fire simulation growth.

Running a simple job on the grid requires many interactions between the different underlying services. This adds extra overhead to the process and increases the time users have to wait for the results. During the application's development phase, the code has to be repeatedly tested for debugging, which makes debugging Grid applications more time consuming.

In order to test our work in a controlled environment with a quicker development cycle, we resorted to the job management tool Ganga[5]. This allows the user to easily swap the endpoint where the job will run, without doing major changes in the submission process. So we increased the flexibility of our systems and introduced code to support for our cluster infrastructure in addition to the grid.

Also, to simplify some tasks in the CROSS-Fire project, we took advantage of the flexibility offered by the operating system and the cluster/grid middleware and made use of scripting to implement an automated task and data management system.

This article describes the developed system, starting with an overview of the distributed computing environment and the associated tools (section 2). Section 3 introduces part of the previous work, which is used for file monitoring and wind/fire simulation. The design considerations and implementation of the system are described in section 4. The document ends with future work considerations, followed by some final conclusions.

2 Distributed Computing environment

The developed platform makes use of cluster and grid resources to run its jobs - a computer program, or a set of programs, scheduled according to some specific rules in a batch system.

The Ganga job management tool offers an abstraction of the cluster, grid or other supported back-end. One of the advantages of Ganga is the possibility of using Python scripts to create and submit jobs in a fully automated way. Other advantage of Ganga is that, due to the abstraction of the back-end, one only needs to describe jobs once. This tool was deployed on the system, increasing the efficiency on the computational resource management task.

Tools

On the grid side, the main tool, currently being used as a base of our software, is the gLite middleware (recently updated to version 3.2)[2], an integrated set of components designed to enable resource sharing on the grid. Each job is described using a high level language, detailing its characteristics and constraints. More information about this subject can be found on the gLite User Guide[8].

[2] gLite homepage: `http://glite.cern.ch/`

The grid is composed by a large set of sites, which share its resources by the users of specific Virtual Organizations (VOs). To ease the task of installing, configuring and maintaining a grid site, a specific software module, named *egee-roll*, was created in the CROSS-Fire project scope[2,4]. This tool is currently being used to manage the UMinho-CP grid site[3], supporting several VOs from different international projects, such as EELA, IBERGRID and ENMR.

On the cluster side, we used the Portable Batch System (PBS), which primary function is to provide job scheduling and allocation of computational tasks among the available computing resources. PBS has a way to specify the files that are to be transferred to the job execution host and the files that are to be obtained upon job completion, using *stagein* and *stageout* options.

3 Previous Work

The platform presented here is supported by a set of tools, previously developed within the project scope. The most relevant building blocks are:

Web Processing Service (WPS) — a set of algorithms that deal with the functionalities of the different components of the CROSS-Fire platform[6].

Console FireStation (CFS) — an interface, based on gvSIG, that provides a rich interaction to the user, including visualization, metadata queries and spatial data queries[6].

Watchdog — an application that allows to check the changes occurred in the content of a file. This tool is useful in the grid context, since it provides access to long jobs' results while they're still running, by sending those changes to a repository, thus opening the possibility of near real-time catch of the data being produced[7].

FireStation — a desktop environment specially designed to the simulation of wildfires, based on wind field and topography models[1]. The fire propagation module was initially "gridified" to allow its execution on the grid. Afterwards, a parallel version of the sequential fire spread algorithm was developed that runs in MPI to accelerate the execution[3].

Simulation applications — Nuatmos and Canyon are two external applications, developed in Fortran, that implement the mathematical wind models with the same name. These models simulate a wind field over some topography.

The Nuatmos model range of application is limited to relatively smooth topographies. However, the solutions are realistic in most cases. The application code is not very time consuming and is numerically stable[1].

The second model, Canyon, is a 3D Navier-Stokes for a generalized coordinate system[1].

[3] UMinho-CP webpage: `https://pop.cp.di.uminho.pt/`

4 The Management System

The CROSS-Fire management system consists of a set of modules (see fig.1), each one performing specific functions:

- **Configuration** - configures the application to adapt it to the running environment
- **Simulation Launcher** - it launches the execution of a job in the cluster or grid
- **Simulation Folder Management** - responsible for data management and storage
- **Simulation Execution Control** - the application that runs tasks and controls its execution

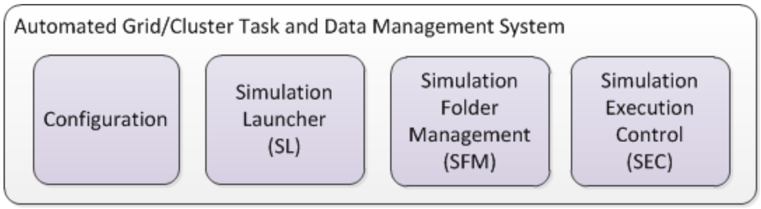
Automated Grid/Cluster Task and Data Management System
Configuration
Simulation Launcher (SL)
Simulation Folder Management (SFM)
Simulation Execution Control (SEC)

Fig. 1. System Modules

4.1 Job Submission Overview

In the CROSS-Fire platform context, a job is an execution of a certain model. Bellow, we describe how the process works (see figure 2):

1. The Console FireStation (CFS) makes a request to the Web Processing Service (WPS);
2. The WPS issues an execution order to the Simulation Folder Management (SFM), passing information about the simulation and the execution, which was received from the CFS. A back-end to execute the job should be chosen, which can be LCG for a grid execution or PBS for a cluster execution;
3. The SFM sends a data download request to the GeoServer;
4. The GeoServer answers the request with the appropriate data;
5. The SFM creates a directory structure, locally and remotely, according to the back-end - in a Storage Element (SE) on grid, or in a disk server on the cluster. Then it carries the necessary data conversions and copies the data into the corresponding locations;
6. The SFM passes the identification of the execution to the Simulation Launcher (SL), that will create a task description;

7. The SL uses Ganga to start a task on the chosen back-end. The application to be executed by the task is the Simulation Execution Control (SEC);
8. When the SEC is scheduled for execution in the grid back-end, it obtains the input data and software sources from the SE, while in a cluster execution, PBS manages to pass the input (*stagein*) data to the execution machine;
9. At the end, the data files and produced meta-data are uploaded to the SE, or grabbed by the PBS, which transfers it (*stageout*) to the disk-server;
10. For the duration of the simulation execution, the Watchdog monitoring tool is continuously sending portions of the produced data to the WPS;
11. The WPS copies the produced data to the AMGA database;

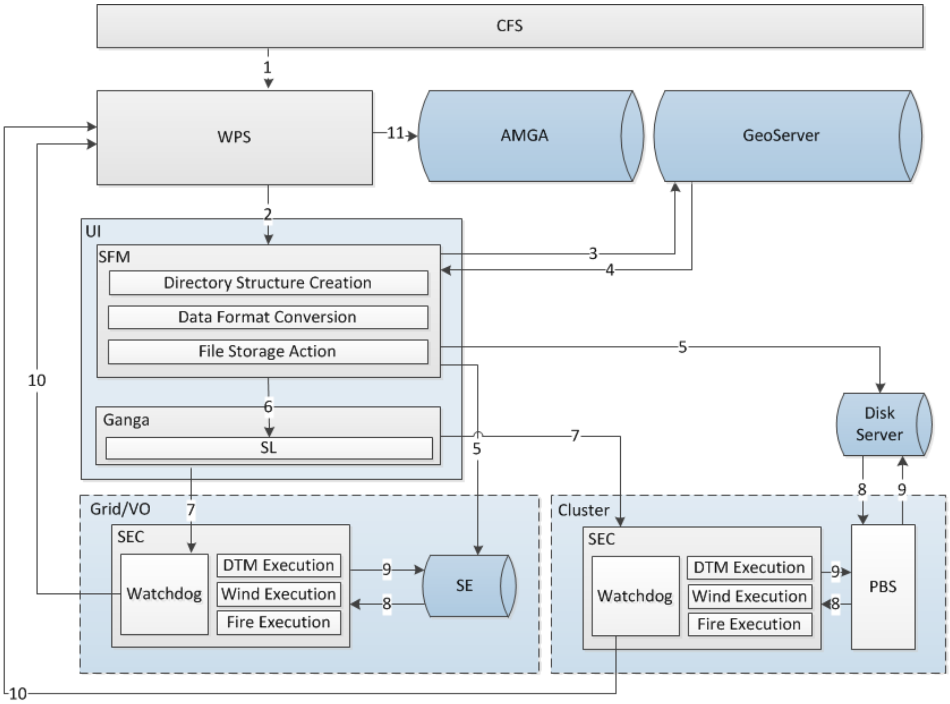

Fig. 2. Job launch description

4.2 Configuration

This module defines a set of variables containing frequently used information about the execution environment. The tables 1 and 2 show the different options available for the two back-ends, cluster and grid execution, respectively.

Table 1. Configuration options for cluster executions

Option	Description
CF_PBS_SE_HOST	disk server
CF_USER	username to access the disk server
PBS_OPTS	batch system extra options
PBS_CF_ROOT	root path in the remote machine
GANGA_PATH	path to the local Ganga installation job files

Table 2. Configuration options for grid executions

Option	Description
CROSSFIRE_SE	SE host
CROSSFIRE_VO	grid virtual organization
LFC_HOST	File Catalog host
CE	computing element
LCG_CF_ROOT	root path in the grid SE
GANGA_PATH	path to the local Ganga installation job files

4.3 Simulation Folder Management (SFM)

A simulation folder is a place where to store the data needed by the executions. This data is the physical representation of the meta-data contained in the AMGA database. The objective of this replication is not only to preserve data for post-mortem analysis, but also to accelerate the access to the data available from past executions[6]. The location of this folder depends on the job execution's back-end system. If the executions are to run on the grid, the data files have to be stored in a grid SE. If the executions are to run in a cluster, the data files have to be stored in a remote disk server.

Data Storage and Upload

The SFM consists of a Bash script and a Java application, working together to manage the data files related to simulations and executions, thus replicating the metadata database. The management of this data comprises three phases: gathering, conversion and storage.

The first phase starts when the SFM obtains the input data. Data descriptions are received from the WPS in a XML data-structure, containing links to topography related data, fetched from the GeoServer, and other data related to the execution, that is to be copied to files on the storage system.

Then, some of the data in XML format needs to be converted to a format suitable to the software that will use it. The fuel description, barrier information, ignition points and meteorology information are also described in the XML data structure that needs to be written in a proper format, to a dedicated file, using a Java Library.

Data storage is made in two different ways, depending on the storage system being used. In the grid, the data is copied from the local machine to the SE using the LCG Data Management tools. In the cluster, the data is copied from the local machine to the remote storage machine using the Secure Copy (SCP) tools.

Data organization

Data is organized in a hierarchical structure, separated by simulations, each one having a distinctive identification. Each simulation folder contains: terrain data, fuel description data, fuel distribution data, a user id, simulation date and description of the simulation area. Each simulation folder may also contain several execution folders, where the executions that belong to the same simulation are stored. Each execution has also a unique identification. Each execution folder contains the grid or cluster job identification, job info, job options, job stdout and stderr, job area, start date, end date and other information related to existent execution algorithms currently supported: fire spread and wind field calculation. In case of a fire spread execution, the folder also contains the ignition points, the barrier points, FireStation control parameters and the identification of the wind field already computed. In case of a wind execution, the execution folder will have precision information, the wind execution model, Nuatmos and Canyon are currently supported, and a file with information about wind conditions, provided by the weather stations existent in the area of simulation.

4.4 Simulation Launcher (SL)

The Simulation Launcher is a Python script used to submit jobs both in the supported back-ends, which provides a simple way of benefit from Ganga features. The type of algorithm to launch can be easily specified through a set of options, described in table 3.

Table 3. Simulation Launcher options

Option	Function
[model]	model(s) to execute in the task
<backend>	back-end where the job will run (PBS for cluster, LCG for grid)
<sim-id>	simulation id number
<model-execution-id>	model execution id number
<set [model-execution-id]>	defines a set of jobs, which belong to the same simulation, that can be launched together. The main job is called a parametric job
<depend>	job dependence definition in PBS back-end jobs

Using these options, it is possible to define the kind of job to be created by Ganga. An example of a command used by the WPS to run an execution is: *submitJob.py –backend=LCG –wind –fire –sim-id=237 –fire-execution-id=55 –wind-execution-id=54*.

The creation of a job via Ganga consists of the definition of the executable to run and the arguments to be passed to that executable, according to the corresponding Job Description Language (JDL). Optionally, the Computing Element (CE) where one wants to run the job may also be specified to allow execution on a controlled environment. The arguments that are passed to the application are listed bellow:

– **Execution type**: type of execution, or executions, to run on the job. Currently, it can be: fire, wind, or both
– **Path to execution**: path to the execution folder in the storage system
– **Path to simulation**: path to the simulation folder in the storage system
– **Path to source files**: path to the source files folder in the storage system
– **Simulation id**: simulation identification number
– **Execution id**: execution identification number

In the case of the grid back-end, we add to the previous list: a) Virtual organization; b) Storage element; c) File Catalog hostname. While some of these arguments are passed as an argument in the SL command, other arguments are read from the configuration file.

To create a job in the LCG back-end, it is only necessary to specify the executable and the respective arguments, while with the PBS cluster back-end, job creation is more complex. In this case, it is also necessary to define the files that are to be copied to the user home (*stagein*) and the files that must be copied from the user home, back to the storage system, when the job terminates (*stageout*).

Another feature of the SL is the capacity to launch parametric jobs. A parametric job causes a set of jobs to be generated from one single Job Description Language (JDL) file. This is useful in cases whenever many similar jobs must be run with different input parameters. Using this feature, Ganga can launch several jobs at the same time, saving time during job submission. Currently, Ganga doesn't support parametric jobs, so, to add this feature to the SL, we used the argument splitter of Ganga. Through this mechanism, one can split the application arguments string field in the job description. So, while a normal job receives a set of arguments, a parametric jobs receives a set of argument sets, where each element of the initial set will be assigned to a sub job application. However, this feature has some limitations. There can be no data dependencies between the jobs that belong to a parametric job, because, currently, there is no way to specify the time execution order of the jobs.

4.5 Simulation Execution Control (SEC)

The Simulation Execution Control is a Bash Script application that controls the execution of a job. It makes decisions, based on the received arguments, to determine which application source code to obtain, compile and execute (figure 3). In addition, it downloads the data files that will be used as input and initiates the Watchdog monitoring tool. This application may be used both in the cluster and in the grid, since the back-end option, specified on the SL, is passed to the SEC.

In the case of a grid execution, the LCG Data Management tools are used to obtain the source code and the input data for the application to run. In the case of a cluster execution, the SEC does not need to obtain any data from any remote machine. The data is obtained through the PBS *stagein* option, which makes the data locally accessible to the user before the job starts so that the SEC may copy it to the execution folder.

The available simulation programs were built in a way that they are continuously writing the produced values in a file, as soon as they are computed, so one can catch the produced values in real-time. In the CROSS-Fire platform, this is done using the Watchdog, which sends the produced values to the WPS. The Watchdog is also used to determine if the job execution terminated without errors. Whenever the execution of the job does not end correctly, the Watchdog sends a signal to the WPS with the word “incomplete”. Otherwise, it sends to the WPS the word “completed”.

At the end of the job execution, the produced data and meta-data must be saved in the storage system. Again, on a grid execution, that information is uploaded to the SE using the LCG Data Management tools. On a cluster execution, data is copied to a user folder. Afterwards, the PBS function *stageout* gathers the output files and copies them to the remote disk server machine. SEC has the capacity of executing several simulation applications in the same job. This means that one can launch a wind simulation, followed by a fire simulation that uses the data produced by the earlier. This feature is useful for two reasons: i) because grid jobs usually take a lot of time to start, one doesn’t have to wait for the conclusion of the first job to launch the second one. ii) Ganga, doesn’t support job dependencies. A simple alternative is to be able to submit several executions within the same job.

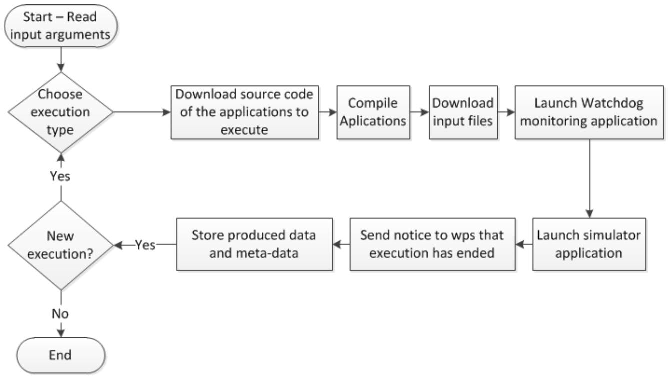
Start – Read input arguments
Choose execution type
Download source code of the applications to execute
Compile Aplications
Download input files
Launch Watchdog monitoring application
Launch simulator application
Send notice to wps that execution has ended
Store produced data and meta-data
Yes
New execution?
Yes
No
End

Fig. 3. Grid Simulation Execution Control workflow

5 Future Work

The current platform needs to be more flexible to be able to integrate new modules, according to the specifications of newer and more sophisticated civil protection requirements.

In order to fulfill this objective, the platform is being enhanced according to the following ideas:

1. Each software module should be delivered in packages, where each package would correspond to an execution model that can be executed in a job. The package would contain not only the source code of the application, but also a description of: i) how to compile the software, ii) the list of the files to download from the storage system before the execution and iii) the list of the files to upload to the storage system after the execution. This could be called Execution Model Packages (EMP).
2. The SEC, could be transformed in a way it could execute any type of module. Currently, the list of files to obtain is described in the SEC. This information could become a part of the EMP.
3. There should exist a SEC for each back-end. Currently, the SEC can be executed both in grid or cluster environments. The separation of this component would ease the development and the addition of new back-ends. It would be the SL's responsibility to choose which SEC to execute in a job.
4. During cluster executions, the input files are copied to the execution folder by the PBS *stagein* and the execution results are collected by the PBS *stageout*. Since the SL is the one that configures the files that are transferred by the PBS *stagein* and *stageout*, the SL must have a way to access the list of input and output files in the EMP.

The ultimate goal is to develop an architecture where one can add new software packages in a simple way.

6 Conclusion

We reported the work we carried out to automate the management of the data and executables within the CROSS-Fire platform, by making use of the tools made available by the cluster/grid middleware.

Each described module started as an independent application. However, as the development advanced, it was necessary to use functions of one module in another module. This raised the need to integrate everything in a single application. As an example of integration, the download of input files in a grid execution is done by the SEC application, but on a cluster execution it is done by the PBS *stagein* and it is the SL responsibility to specify which files to transfer to the execution folder.

Looking back, there are some loose ends that could be changed. The reason for this is that, during the development phase, the CROSS-Fire team gained a greater knowledge of the used tools. This knowledge can be applied in future iterations of this project.

Acknowledgements

This research was funded by the Portuguese FCT through the CROSS-Fire project. It also benefited from UMinho's participation on EC FP7 E-science grid facility for Europe and Latin America (EELA2) and, more recently, in the EC FP7 Grid Initiatives for e-Science virtual communities in Europe and Latin America (GISELA).

References

1. A. Lopes, M. Cruz, D. V. An integrated software system for the numerical simulation of fire spread on complex topography. Environmental Modelling Software, Volume 17, Issue 3, 2002, p. 269-285.
2. A. Pina, B. Oliveira, A. Serrano, V. Oliveira. EGEE Site Deployment & Management Using the Rocks toolkit. In *Ibergrid: 2nd Iberian Grid Infrastructure Conference Procedings* (2008), Silva, F and Barreira, G and Ribeiro, L, Ed., pp. 285–295. 2nd Iberian Grid Infrastructure Conference (Ibergrid 2008), Porto, Portugal, May 12-14, 2008.
3. A. Pina, R. Marques, B .Oliveira. FireStation: From Sequential to EGEE-Grid. Proceedings of the first EELA-2 Conference, Bogotá, Colombia, CIEMAT, February 2009.
4. B. Oliveira, A. Pina,A. Proenca. EGEE site administration made easy. In *Ibergrid: 4th Iberian Grid Infrastructure Conference Proceedings* (2010), Proenca, A and Pina, A and Tobio, JG and Ribeiro, L, Ed., pp. 295–306. 4th Iberian Grid Infrastructure Conference (Ibergrid 2010), Braga, Portugal, May 24-27, 2010.
5. F. Brochu, J. Ebke, U. Egede, J. Elmsheuser, K. Harrison, H. C. Lee, D. Liko, A. Maier, A. Muraru, G. N. Patrick, K. Pajchel, W. Reece, B. H. Samset, M. W. Slater, A. Soroko, C. L. Tan, D. C. Vanderster, M. Williams. Ganga: a tool for computational-task management and easy access to Grid resources. Computer Physics Communications, Volume 180, Issue 11, p. 2303-2316 .
6. Pina, A., Oliveira, B., Puga, J., Esteves, A., and Proenca, A. A platform to support Civil Protection applications on the GRID. In *Ibergrid: 4th Iberian Grid Infrastructure Conference Proceedings* (2010), Proenca, A and Pina, A and Tobio, JG and Ribeiro, L, Ed., Netbiblo, pp. 355–367. Braga, Portugal, May 24-27, 2010.
7. R. Bruno, B. Barbera, E. Ingrà. Watchdog: A job monitoring solution inside the EELA-2 Infrastructure.
8. S. Burke, S. Campana, E. Lanciotti, P. Lorenzo, V. Miccio, C. Nater, R. Santinelli, A. Sciabà . Glite 3.1 User Guide.