CHARON System–Framework for Applications and Jobs Management in Grid Environment

Jan Kmuníček,[1,2] Petr Kulhánek,[1,3] and Martin Petřek[1,3]

1. CESNET z.s.p.o., Zikova 4, CZ-16000 Praha, Czech Republic
2. Institute of Computer Science, Masaryk University, Botanická 68a, CZ-60200 Brno, Czech Republic
3. National Centre for Biomolecular Research, Faculty of Science, Masaryk University, Kotlářská 2, CZ-61137 Brno, Czech Republic

Abstract

We present a generic system for utilization of application programs in the EGEE Grid environment–the CHARON system. Charon was developed by computational chemistry community in the Czech Republic to provide easily manageable, comfortable, and modular environment to fulfill specific requirements of computational chemistry application users. It currently offers an alternative to standard LCG/EGEE UI environment in application-generic Virtual Organization for Central Europe (VOCE). Present-day implementation of Charon system is completely compatible with scripting roots of the EGEE Grid environment, provides comfort computational jobs management by encapsulation of available LCG/EGEE middleware environment, support for smooth administration of large amount of computational jobs and also enables easy retracing of already finished calculations. Compared to widespread graphical user interfaces (*e.g.* portals) Charon is oriented towards users requiring simple but feature-rich and powerful command line and scripting interface that offers support for tens to hundreds of jobs within a single research project. Taking into account the cornerstones on which the system is built - modularity and generality - it is not targeted only for molecular modeling purposes but it also represents a generic application framework easily adaptable for broad set of generic application areas and their specific programs. Moreover Charon also permits utilization of resources from non-EGEE Grids. Therefore it is expected to be one of useful tools available for promoting usage of Grid environments for general purposes.

1 Introduction

Job submission and its subsequent management are crucial tasks for successful utilization of cluster and/or grid environments that are controlled by various batch systems (PBSPro,[1] OpenPBS,[2] LSF,[3] scheduling components of grid middleware such as Globus,[4] LCG,[5] gLite,[6] and others). Each batch system has unique tools and different philosophy of its utilization. Moreover, the provided tools are quite raw and users have to perform many additional tasks to use computer resources properly. Since end-users of clusters and/or grids are more interested in some particular scientific problems, the difficult use of batch system might have a negative impact on the efficiency of resource utilizations and therefore on studied scientific problems. The introduced Charon System represents one possible solution of problems associated with the utilization of low-level batch system commands.

Charon System is the heart of the Charon Extension Layer[7] (CEL). The Charon Extension Layer is a command line interface (CLI) that consists from two subsystems: Module System and Charon System (Fig. 1). Module system is used for the management of installed application programs. It solves problems connected with execution of applications on machines with different hardware or operation systems and it is also able to simply execute applications in parallel environments. Charon System is a specific application managed by Module System, which introduces a complete solution for job submissions and managements. These two parts form together a unique and consistent solution not only for job submission and management but also for easy job preparations.

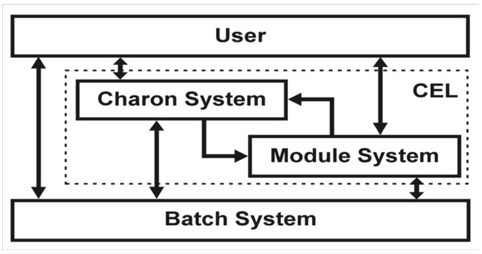

Fig. 1: Charon Extension Layer built from Module System and Charon System.

2 Charon Extension Layer

2.1 Module System

The utilization of application software does not introduce any problems till different versions of the same application are required to be installed on the same host or if two different applications provide programs with the same name. Described situation is very common, especially in scientific community, and there have to be used some tools solving aforementioned problems. Well known solution is *Environment Modules Project* (EMP).[8] This project provides tools for the dynamic modification of a user's environment via modulefiles. The modulefile contains the information needed to configure the shell for an activated application. However, the solution provided by EMP has several drawbacks. It is very difficult to solve problems with the same application that was built for different target architectures or operation systems. The second problem appears if parallel execution is also required. These issues are usually solved by different modulefiles, which describe particular cases. This approach solves the problem, but it is very confusing for users. We developed Module System, which uses basic ideas from EMP, but in addition solves all the drawbacks mentioned above.

Module System introduces abstract services for the management of application software within CEL. Each application software is described by a module in Module System. The module contains all information that is necessary for work with a particular software (*e.g.* PATH setup, additional environment setup, *etc.*). This information is provided for all application's builds including different versions or target architectures or parallel environments. This information is addressed by module name. The module name consists of four parts describing: application's name, application's version, target architecture, and parallel environment for application execution. The module name containing all these four parts uniquely describes one particular application build (so-called realization). Examples are shown in Tab. 1.

Full Module Name (Realization)	***Description***
amber:8.0:xeon:shmem	Amber simulation package, version 8.0, optimized for Intel Xeon processors, parallel execution is enabled *via* shared memory device of mpich library.
turbomole:5.6:i686:single	Turbomole package, version 5.6, optimized for i686 architecture with sequential execution.

Tab. 1: Examples of names of Module Realizations.

Up to this point, EMP approach and Module System approach are similar. However, the separation of module name into four parts enables only selected subparts of module name to be specified by an end-user. The missing parts are then automatically selected by Module System from either default (predefined) values or from result of automatic procedures, which selects the best existing application realization for provided computational resources. This procedure is shown in Fig. 2. Described module naming technique brings excellent flexibility of system. If the users specified only application name (and optionally application version) in their job scripts, the very same script can be used without any modification on various platforms or in sequential or parallel execution. Information about currently accessible resources is partially provided by Module System itself and partially by user through Charon System at the time of job submission (this will be described later in chapter 2.2).

amber	=>	amber:8.0:auto:auto	=>	amber:8.0:xeon:single
(user specification)		(completion by default setup)		(final name resolved according to current available computational resources)

Fig. 2: Example of module name completion.

Module System provides several commands. Aside from several administrative commands, two commands are important from the user point of view. **module** command provides both informational and executive services. It is able to list available and activated applications and, moreover, further information about Module System. It also modifies shell environment if user wants to activate specified applications. The second command is **modview**. This command alters the setup of Module System used in informational services, *e.g.* user can change output colors of printed information and other items.

Module configuration information is stored internally in XML format that makes setup of Module System very straightforward. An example of such file for Charon System is shown in Fig. 3. Module System is able to modify shell environmental variables (set, prepend and append modes for variables containing items separated by colon) or execute scripts during loading or unloading module. Moreover information about possible conflicts or dependencies between modules might be used in configuration file. Module System does not use these files directly, instead, they are cached into one file, which is then finally processed by the system.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- charon module config -->
<module name="charon">
        <realization ver="0.8" arch="i686" par="single">
                <variable name="CEL_PACKAGE_DIR"
                          value="$SOFTREPO/charon/0.8/i686/single"
                          operation="set" priority="modaction"/>
                <variable name="CEL_STAMP_VALUE"
                          value="charon:0"
                          operation="set" priority="modaction"/>
                <variable name="PATH"
                          value="$SOFTREPO/charon/0.8/i686/single/bin"
                          operation="prepend"/>
                <variable name="CHARON_ROOT"
                          value="$SOFTREPO/charon/0.8/i686/single"
                          operation="set"/>
                <script   name="$SOFTREPO/charon/0.8/i686/single/bin/_charon_init"
                          type="inline"/>
        </realization>
        <default ver="0.8" arch="auto" par="auto"/>
</module>
```

Fig. 3: Example of module configuration file for Charon System.

Other configuration files in XML format are used for additional settings. One of them contains translation rules among various target architecture names (*e.g.* rules describing compatibility among various architectures) and rules for choice of parallel mode names if parallel execution is possible ($N_{CPU} > 1$). For information purposes, Module System uses configuration file with the specification of categories to which applications belong. This file also contains the setup for the visualizations of list of available and activated applications (colors, section delimiters, *etc.*).

All the above-mentioned things work nicely in environments that are able to operate with shared volumes over all worker nodes. This architecture is usually available in smaller grid or cluster environments. Unfortunately, this is not feasible in larger grids because sub-sites are well separated and only limited number of services can be used to inter-operate among them. In the latter case, Module System had to be extended in such a way that these problems were eliminated. New feature was added to the system: modactions. A modaction represents the execution of script when predefined action is performed with **module** command. Each action can have its own modaction that is determined in appropriate configuration file.

Modaction script for module activation was used to solve above discussed problem. This script behaves differently on user interface (UI), on which it does not perform any action because Module System can operate with realizations stored on local file system, and on worker node (WN). On worker node, it checks if activated realization has been already installed in temporary software repository. If not, it downloads particular package from storage element and installs it to this repository. Set of accompanied administrative commands was written. These commands use software repository on local UI file system to build the mirrored version stored on storage element. The last thing that had to be solved was the installation of Module System itself on worker node. This task is performed with wrapper script of Charon System that downloads package containing core services–commands, module databases and configuration files and initializes them before user's script execution.

2.2 Charon System

Charon System is used for computational job submission and its subsequent management by an end-user. To simplify job submission as much as possible, the solution of two problems was determined to be very important. Firstly, common user is not happy to specify files that form input or output sandbox. The latter is especially crucial because if some important file is omitted in that specification then the result of such calculation is meaningless even if the calculation succeeds. Secondly, the user preferably does not want to remember complicated specification of computational resources.

The first described problem is solved by Charon System in such a way that all job data have to be in one directory–the job directory. Charon System then considers all files in the job directory (recursively) as input files and when calculation is finished all data in the job directory are considered as results (also recursively). In the job directory, at least one shell script has to exist. Charon System then executes this script on worker node. The second problem is solved with aliases. User can define by **palias** command alias that specifies all required resources and properties of worker nodes. The name of alias is then used during job submission. Examples of aliases are shown in Tab. 2.

PBS batch system

alias	*queue/VO*	*syncmode*	*properties*	*resources*
nfast	ncbr	sync	lcc#xeon	-
mopac	ncbr	sync	-	nodes=perian16.ics.muni.cz:ppn=1:per#p3

LCG middleware

alias	*queue/VO*	*syncmode*	*properties*	*resources*
skurut	voce	gridcopy	-	skurut17.cesnet.cz:2119/jobmanager-lcgpbs-voce

Tab. 2: Examples of aliases.

Each alias can describe required queue (PBS batch system) or virtual organization (LCG middleware), syncmode, properties, and resources. *Properties* describe worker node features for PBS batch system. In the case of LCG middleware, they are equivalent to *requirement* item from Job Description Language (JDL). In comparison with properties, *resources* determine more precisely required worker nodes. In above examples, mopac alias forces execution on worker node perian16.ics.muni.cz. This is used for jobs that use mopac application because it is licensed only for this particular node. In the scope of LCG middleware, resources determine target computing element.

An important item of alias definition is *syncmode*. This item determines how Charon System operates with input and output sandboxes (*e.g.* with contents of job directory). Current implementation recognizes following syncmodes for PBS-like batch systems: local, sync, nosync, and nocopy. *local* is very special syncmode because it also specifies resources in such a way that job is executed on the same host and in the same directory as job input directory. This is useful if large amount of data has to be analyzed. User copies that data to scratch directory on free worker node and then it starts job on such node. This mode is valuable only on small clusters because on larger sites it is problem to find free worker node. *sync* mode copies the contents of job input directory to worker node and after job execution the contents of job directory on that node is copied back. In *nosync* mode, the backward movement of data is not performed. *nocopy* mode does not copy data between job input host and worker nodes. In this case, it is required that input job directory is on volume which is also accessible from worker nodes. *nocopy* mode is mainly used for parallel execution of some chemical applications which requires shared data among all processes. The situation on sites utilizing LCG middleware is less complicated. In that case, Charon System uses only two syncmodes: gridcopy and stdout. *gridcopy* is equivalent to *sync* mode but data transfer is mediated through storage element. *stdout* copies content of job input directory through storage element to worker node and when a job is terminated only file that contains standard output from job script executions is copied back.

Job submission is performed by **psubmit** command. This command has two mandatory parameters. First parameter specifies required resources. User can specify either queue/VO or existing alias. The second parameter is the name of job script or job input file if automatic detection of job type is enabled. Two additional parameters are optional. The third parameter determines number of required CPUs and the fourth overwrites syncmode taken from default setup or from alias. Example of job script and its submission is shown in Fig. 4.

```
[jobdir]$ ls
  md_test  prod001.crd  prod.in  solv.top

[jobdir]$ cat md_test
  # example of MD simulation
  # activate AMBER package
  module add amber
  # do MD simulation with sander module of AMBER
  sander -i prod.in -p solv.top -c prod001.crd \
         -o prod.out -r prod.rst -x prod.traj

[jobdir]$ psubmit voce md_test
```

Fig. 4: Example of job submission.

It is important to repeat here that if parallel execution of sander program is required in the aforementioned job then only one item has to be changed. This is the number of required CPUs provided as the third parameter of **psubmit** command.

Charon System creates several runtime files in job directory which can be further used in job monitoring. For these purposes, **pinfo** and **pgo** commands can be used. **pinfo** command shows current status of job, *e.g.* when job was submitted, started, and finished. The output also contains summary of required resources at the job submission time. **pgo** command can be used only on clusters using PBS-like batch systems. This command will log user on worker node and change current directory to job directory if job is running or if job was terminated but job results were left on worker node (*nosync* mode). This is very useful because the user is able to monitor the job progress on-line.

When the job is terminated then the results have to be obtained. *sync* mode automatically copies results back. For *gridsync* and *stdout* modes, this has to be performed explicitly by **psync** command.

3 Sites Using Charon

Charon Extension Layer is installed on several clusters and grids. There are installations on two small clusters at the National Centre for Biomolecular Research.[9] These clusters are used by Laboratory of Computational Chemistry for postprocessing of data obtained on other larger computational facilities. These are METACentrum[10] and Virtual Organization for Central Europe (VOCE).[11] Charon Extension Layer provides the same interface for job submission among these sites. This enables easy utilizations of resources provided by completely different architectures.

METACentrum covers majority of activities concerning super-, cluster- and grid computing and/or high performance computing in general in the Czech Republic. The aim of the METACentrum project is management of current computational resources and also their extension, in cooperation with the largest academic computing centers in the Czech Republic. At the end of 2005, the computational resources available in METACentrum range from PC clusters to SMP systems and provide about 500 CPUs.

VOCE is a computing service, which has been established in order to directly support Central European researchers' needs in the area of Grid computing. The Central European region is currently formed by Austria, Czech Republic, Hungary, Poland, Slovakia and Slovenia as defined in the EGEE Technical Annex. At the end of 2005, the computational resources available in VOCE are equivalent to about 500 CPUs.

4 Application Software

We are building Common Software Repository for all covered sites. We try to keep this repository the same as possible on all sites. However there are some differences, which mainly result from license restrictions. Partial list of applications in Common Software Repository is shown in Tab. 3.

Application	***META***	***VOCE***	***Description***
amber[12]	I	I	molecular dynamics simulations
autodock[13]	I	I	docking of flexible ligands to static proteins
babel[14]	I	I	conversion program among various chemical structural formats
caver[15]	P	I	analysis of access pathways to active sites in proteins
cpmd[16]	I	T	*ab initio* Car-Parrinello molecular dynamics
dalton[17]	I	T	quantum mechanics calculations
delphi[18]	I	I	numerical solutions of the Poisson-Boltzmann equation
gaussian[19]	I	-	quantum mechanics calculations
gnuplot[20]	I	I	scientific data and function plotting utility
gromacs[21]	I	P	molecular dynamics simulations
qhull[22]	I	I	calculations of convex hull, Delaunay triangulation, *etc.*
mopac[23]	I	P	semiempirical quantum mechanics calculations
molscript[24]	P	I	visualizations of molecular structures
povray[25]	P	I	rendering program
turbomole[26]	I	I	quantum mechanics code optimized for Intel architecture

Tab. 3: Partial list of applications managed by Module System.
I – installed; P – planned; T – in testing;

5 Conclusion

Charon Extension Layer presents a generic, uniform, and modular approach for job submission and management in a wide range of grid environments. It has been successfully implemented and tested in VOCE (Virtual Organization for Central Europe) using LCG middleware, in METACentrum (Czech national grid project) using PBSPro batch system, and in sets of local PC clusters using OpenPBS batch system. It consists of two co-operating subsystems: Module System and Charon System. Their tight interconnection enables to provide following set of main features: encapsulation of a single computational job; minimization of overhead resulting from direct middleware use (JDL file preparation, *etc.*); an easy submission and navigation during job lifetime; powerful software management together with comfortable enlargement of available application portfolio; and fast innovation in the development of new computational methods and techniques. Moreover, CEL does not restrict the utilization of the native grid midleware and/or web based approaches in any way.

Acknowledgments. Financial support from the Ministry of Education, Youth, and Physical Training of the Czech Republic (contract number MSM0021622413) and from European Commission (the EGEE project - contract number IST-2003-508833) is gratefully acknowledged. We also thank to Luděk Matyska and Jaroslav Koča for their valuable advices and support.

References

1. PBSPro, http://www.altair.com/software/pbspro.htm
2. OpenPBS, http://www.openpbs.org/
3. LSF, http://www.platform.com/Products/Platform.LSF.Family/home.htm
4. Globus, http://www.globus.org/
5. LCG, http://lcg.web.cern.ch/LCG/
6. gLite, http://glite.web.cern.ch/glite/
7. Charon Extension Layer, http://egee.cesnet.cz/en/voce/Charon.html
8. Environment Modules Project, http://modules.sourceforge.net/
9. NCBR, http://www.ncbr.chemi.muni.cz/
10. METACentrum, http://meta.cesnet.cz/
11. VOCE, http://egee.cesnet.cz/en/voce/index.html
12. Amber, http://amber.scripps.edu/
13. AutoDock, http://www.scripps.edu/mb/olson/doc/autodock/
14. Babel, http://smog.com/chem/babel/files/
15. CAVER, http://viper.chemi.muni.cz/caver/
16. CPMD, http://www.cpmd.org/
17. Dalton, http://www.kjemi.uio.no/software/dalton/dalton.html
18. DelPhi, http://trantor.bioc.columbia.edu/delphi/
19. Gaussian, http://www.gaussian.com/
20. Gnuplot, http://www.gnuplot.info/
21. Gromacs, http://www.gromacs.org/
22. Qhull, http://www.qhull.org/
23. Mopac, http://www.cachesoftware.com/mopac/index.shtml
24. Molscript, http://www.avatar.se/molscript/
25. POV-Ray, http://www.povray.org/
26. Turbomole, http://www.cosmologic.de/turbomole.html