A Framework for Execution of Computational Chemistry Codes in Grid Environments

André Severo Pereira Gomes[1], Andre Merzky[2], and Lucas Visscher[1]

[1] Department of Theoretical Chemistry, Faculty of Exact Sciences Vrije Universiteit Amsterdam, De Boelelaan 1083, 1081 HV Amsterdam The Netherlands
[2] Department of Computer Science, Faculty of Exact Sciences Vrije Universiteit Amsterdam, De Boelelaan 1081A, 1081 HV Amsterdam The Netherlands
a.gomes@few.vu.nl, merzky@cs.vu.nl, visscher@chem.vu.nl

Abstract. Grid computing is a promising technology for computational chemistry, due to the large volume of calculations involved in appplications such as molecular modeling, thermochemistry and other types of systematic studies. Difficulties in using computational chemistry codes in grid environments arise, however, from the fact that the application software is complex, requiring substantial effort to be installed on different platforms. Morever, these codes depend upon task–dependent sets of data files to be present at the execution nodes. Aiming to improve the usability of different quantum chemistry codes in the distributed, heterogeneous environments found in computational grids, we describe a framework capable of handling the execution of different codes on different platforms. This framework can be divided into three independent parts, one dealing with the mapping of a calculation to a set of codes and the construction of execution environments, one dealing with the management of grid resources, and one that takes care of the heterogeneity of the environment. The suitability of this framework to tackle typical quantum chemistry calculations is discussed and illustrated by a model application.

Keywords: Grid Computing, Computational Chemistry, Grid(lab) Application Toolkit, Heterogeneous Environment, Command–line Interface.

1 Introduction

Grid computing has evolved rapidly over the past few years, and has reached a stage where scientists outside the field of computer science are beginning to explore the potential benefits of these new technologies. Computational chemistry is an application area where grid computing may have a significant impact, due to its ever–increasing demand for computational resources. While grid computing may not be suitable for all problems taken up by computational chemists, due to memory or other hardware requirements, it will likely be very useful in increasing throughput in cases where calculations are not truly expensive in themselves but are to be performed in large numbers. Examples of these are potential energy surface (PES) scans, accurate thermochemical calculations, combinatorial design of new compounds, or in finite temperature statistical averaging of molecular properties.

The relatively few initiatives to use grid computing in computational chemistry have so far involved either the modification of an existing code to use some of the functionality of a grid middleware[1, 2], such as Globus[3], or the creation of portals[4, 5], user interfaces and/or specialized middleware to handle both grid–related tasks, such as authentication, as well as application–specific tasks, such as the creation of inputs and visualization of results[6, 7, 8]. These efforts, typically restricted to one or a few codes, have a significant drawback because they rely on a specific middleware to access grid resources (some projects use the Globus toolkit while other use UNICORE[9]), demanding major revision as the middleware evolves. It is not clear, moreover, how easy it would be in these approaches to support different codes, particularly in heterogeneous environments. It is also not clear at this stage how to achieve high throughput in environments that rely solely on graphical interfaces to prepare and/or submit jobs.

Having these issues in mind, in this paper we describe a framework that allows the use of grid resources by computational chemists, which: (a) already has support for different codes and is easily extendable to include new application software or support more complex work flows ; (b) allows for use of heterogeneous grids; (c) has no dependence on a specific middleware; and (d) has a simple user interface. A standard command–line tool is supplied, that can be replaced by graphical user interfaces or grid portals if desirable.

2 Implementation Details

The framework constructed can be divided in four individual components which are responsible for: (a) handling the use of binary executables and platform–dependent data, in order to allow for the use of grid environments with varying degrees of hardware and/or software heterogeneity; (b) performing tasks related to the grid environment, such as requesting resources, scheduling jobs for execution, cancelling jobs, verifying a job's status, transferring files and so on; and (c) creating jobs (which in this context are characterized by the collection of the input and eventual restart data for a calculation with a program, or collection of programs). These are implemented in the Python programming language, and are integrated via a basic command–line user iterface.

2.1 User Interface

The standard interface between the user and the stages of job creation, submission and management is a command–line tool, with which it is possible to create, submit, cancel or query information from jobs running on grid resources. Job creation does not depend on the other actions and needs only little information about the grid environment that will be used. The other actions inherently require more knowledge about the specific grid and middleware deployments.

In an interactive procedure the user is asked to first specify some global preferences for hardware resources(minimal memory requirements, preferred type of architecture, etc.), software resources (computational chemistry code to be used) and standard location of the input files. This step is typically done only once and serves to restrict the amount of data that is to be given in the production stage. The global preferences may later be adjusted by creating a *project* that defines a label for a set of specific preferences for hardware, software and location that is optimal for the type of work that is to be carried out. Also this configuration step is done interactively, and is only necessary when a new project is created or when the user want to adjusts its defaults. In a given project one may then easily define many *jobs* by giving only the specific input and workflow for the individual tasks. One may thereby combine job creation and submission if desired.

2.2 Handling Platform–Dependent Data

An important issue one faces when trying to use computational chemistry codes, particularly in heterogeneous grid enviroments, is how to select an adequate executable. While in some fields it might be possible to generate the correct binary from source code at the execution host, this is impractical for computational chemistry codes, that tend do be fairly large (a common figure is 500k lines or larger of FORTRAN code). In such a case an"on the fly" compilation can easily dominate the total CPU time used. Moreover, most codes are difficult to compile without user intervention, because they usually contain legacy (and sometimes poor quality) code and usually require special mathematical libraries that may not be available at the execution host.

Because of these difficulties, we have opted to handle heterogeneous environments by precompiling different binaries for different combinations of operating systems and machine architectures that are made available at one or more remote locations via `http` or `gridftp`. Upon execution, the job will determine the operating system and architecture of the execution host and retrieve the appropriate binary (in compressed form), using tools like globus' file transfer facilities, or other transfer mechanisms, such as `curl` or `scp`, depending upon the way the gereral or project preferences were set in the configure step(s).

One problem that can not be solved at the moment is the fact that commercial codes often have a licensing scheme that is incompatible with use in computational grids that connect more than one institution. ADF[10], for instance, has a licensing scheme that verifies at runtime the place of execution, and will stop if no valid license file is found. Obtaining a license that enables such codes to be used within the whole grid infrastructure is too expensive in the current setup, while vendors will be reluctant to relax the license checking because it will increase the possiblity of unauthorized use. Before grids can be widely employed in production work it is thus necessary that vendors and middleware developers come up with more suitable licensing schemes, e.g. based on a "pay per use" system.

2.3 Use and Managementen of Grid Resources

Interaction with grid resources, i.e. for the transfer of files, submission of jobs, cancellation of jobs, resource management and so on, is handled by the Grid Application Toolkit (GAT) [11], developed in the GridLab project. This toolkit was chosen since it provides several attractive features, such as: (a) an API that closely resembles UNIX system calls, which makes the function calls to the corresponding middleware's functionality easy to understand and relatively simple to use, (b) the freedom to easily change grid middleware if necessary for a particular case or grid deployment; (c) a built–in persistent database, referred to as the advert service, which keeps track of jobs' status, the location of files, and execution hosts; and (d) the availability of wrappers in Python to the C reference implementation. The use of Python wrappers makes it easier to maintain the code and, more importantly, to later reuse components in other projects within our group, such as graphical user interfaces.

2.4 Job Creation

The job creation module gathers all input file(s) for a given computational code and uses the information contained in these file(s) to generate a script for the excution of this job. In order to do so a two–step procedure for job creation was implemented. In the first step, all data to be transferred (such as execution scripts, input files, special basis sets, and other non–executable data) is collected in a tar–archive. This archive is included in the script during its assembly on the second step. This script, which encapsulates all the remotely executed commands, then acts as an application manager–like "container" for the grid–unaware codes such as those generally used in chemistry.

The basic operations involved in the first step are: (a) scan the input file(s) to determine the data dependencies that should be met (required basis set files, libraries, etc); (b) collect all required data files in a temporary directory; and (c) create a compressed tar file from the directory and convert this to an encoded ASCII format via utilities such as `uuencode`; (d) remove the temporary directory. In the implementation we minimized the amount of data to be archived, so that the data transfer to a remote host should take little time, also on slow networks. In line with this philosophy, we avoid including executable binaries in the container script, so that it is possible to handle large numbers of jobs without unnecessary replication of data on the submission host. We are currently working to incorporate more complex workflows that allow for use different codes at the remote host. In this case, the input is a "control file" specifying inputs for each calculation, and the actions and dependencies that connect the individual calculations.

In the second step a Bourne shell script is constructed automatically, taking into account special demands of the code, or collection of codes, to be used. The encoded archive is inserted into this script and, upon execution at the remote host, decoded, uncompressed and unarchived into a directory private to the job. This "sandboxing" ensure that calculations will not interfere with other jobs that may be executing at the same host. Moreover, it also allows the location of output and intermediate files (restart data etc) to be easily determined on the remote host, so that GAT's functionality may be used to inspect and copy files to/from the remote host during a calculation.

By using the "container" approach, which is in fact an acknowledged practice in grid environments[12, 13], it is possible in principle to execute any existing code running under UNIX–like environments invoked for calculation as a command–line application (either directly or with the help of shell scripts). As this situation is found in most of the computational chemistry codes available to date, this procedure can handle the execution of any code in platforms for which these are supported. Currently ADF[10], Dalton[14], Dirac[15] and Dacapo[16] are supported, but since code–dependent actions are largely confined to scanning and setting shell environment variables, it is relatively easy to support additional codes within this framework. A second feature that makes this approach attractive is the control over the moment when binaries and other data are fetched, so that these actions could be postponed until they are actually needed, thus allowing for a more balanced communication overhead.

3 Application Tests

The framework introduced here has been largely developed and tested for the currently supported codes on the DAS–2 computer[17], a distributed but architecturally homogeneous Linux machine which is part of the Dutch Grid initiative. However, as one of its strengths lies in handling heteronegeous systems, we also present some results from calculations run on the GridLab testbed[18] and on a small–scale test grid assembled within our Theoretical Chemistry Department (TC–VU) using Macintosh desktop computers as execution hosts and the Xgrid tool as the middleware[19].

Since we are concerned with evaluating the framework, we have simply taken representative examples from the test sets supplied with the quantum chemistry codes and executed them at the platforms mentioned above. Important information that was to be gained in these tests is the effect of communication overhead, due to transferring of files to remote locations, on the total execution time. Table 1 summarizes the timing results obtained for ADF, Dirac and Dalton job performed at different geographical locations. The binary repository is located within the Theoretical Chemistry Department network and, apart from calculations performed at the departmental grid, the DAS–2 head node at the Vrije Universiteit (`fs0.das2.cs.vu.nl`) is used as a submission host.

From the results it is clear that, even for the short jobs considered, the amount of time spent on executing a given computational chemistry code dominates the total execution time for the job. Moreover, since the actions related to setting the execution environment on the remote hosts, such as transferring binary files, appear to take a roughly constant time for a given machine and type of calculation, it is reasonable to assume that already for moderately long calculations the grid-related overhead involved will become negligible.

The role of high–speed interconnection between grid locations is also evident from the results. Very similar results are obtained for the departmental grid

Table 1. Total job wall times (in seconds) and the total calculation/total job time ratios for ADF, Dalton and Dirac test runs in different grid environments. Here *TC VU* denotes the departmental grid; *DAS2 VU* and *DAS2 UU* nodes within DAS–2 (`fs0.das2.cs.vu.nl` and `fs4.das2.phys.uu.nl`); and *Gt PL* nodes within the Gridlab testbed (`eltoro.icis.pcz.pl`).

		TC VU		*DAS2 VU*		*DAS2 UU*		*Gt PL*	
Code	Test	time	ratio	time	ratio	time	ratio	time	ratio
ADF	C2H4 TDCDFT	154	.92	391	.95	388	.94	259	.81
	EPR-SOO	339	.91	337	.92	325	.94	207	.71
	NMR VOCl3	364	.91	597	.95	589	.96	387	.83
	PF3 Nmr	380	.94	824	.97	822	.97	560	.88
	VO Collinear	506	.96	775	.98	782	.98	1264	.96
Dalton	prop vibana	455	.96	568	.99	587	.99	770	.96
	walk solvmag	206	.94	563	.99	568	.99	713	.96
	cc e triplet	189	.92	236	.97	215	.97	195	.88
	dft qr nosym	202	.90	375	.97	362	.98	509	.94
Dirac	b3lyp ch4	251	.99	527	.994	524	.994	778	.99
	zora xe	48	.94	90	.97	92	.97	176	.93
	rhonuc co	283	.99	711	.996	706	.996	977	.99
	135TCC geomopt	461	.99	980	.997	968	.997	1225	.99

(whose nodes sit on the same high–speed local network) and different DAS–2 locations (which are connected through the high–speed SURFnet[20] network), while a degradation in performance, seen in the decrease of percentage of the total time spent on the actual calculations, is visible for the Polish host in the Gridlab testbed. Such degradation is predominantly due to the longer times spent on transferring executables. While this may make the use of this framework impractical for very short calculations (defined as taking less than five minutes on a single CPU), most chemical calculations required for production work take considerably longer, so that it is effective to also use geographically distant hosts. Short jobs that use the same executable could furthermore be combined into a single job, to reduce the transfer/total execution time ratio in that case as well.

4 Conclusions

We have discussed a framework that allows for the use of different quantum chemistry codes in heterogeneous grid environments. This framework is composed of three interacting but formally independent parts, integrated via a command–line interface: the first handling application–specific task of creating an execution script for a given type of calculation; the second handling grid–related tasks, such as submission, execution and retrieval of results; and the third comprising the infrastructure that enables performing calculations in an heterogeneous environment.

The strategy used in creating this framework consists of using components that are available for UNIX–like or Windows–based installations, namely the Python programming Language and the Grid Application Toolkit (GAT). The execution of computational chemistry codes at this time requires the availability of the Bourne shell, and was only tested in UNIX–like systems, but it should be possible to have that available on Windows–based systems, or eventually use the upcoming Microsoft shell (MSH)[21] as a replacement of the Bourne shell in these systems.

In terms of functionality for computational chemistry tasks, the current implementation of this framework provides enough flexibility for the execution of one or more calculations as a single job, irrespective of the codes involved in calculating each step. Given the successful use of this implementation in pilot calculations with the ADF, Dalton, Dirac and Dacapo codes, we are currently working to incorporate other codes into this setup, as well as testing this framework in a production environment.

Acknowledgements

ASPG and LV acknowledge financial support from National Foundation for Computing Facilities (NCF) through the NCF research grant number NRG-2004-05, and technical support from SCM (Scientific Computing and Modeling) via Dr. Stan van Gisbergen and Dr. Olivier Visser regarding different aspects of the use of ADF in this project. Discussions with Dr. Drew McCormack, particularly regarding the use of Apple's Xgrid framework and the setup of the department–wise test grid are also acknowledged. AM acknowledges the support of the GridLab project and the GAT development team, in particular from Hartmut Kaiser, who provided the Python language bindings.

References

1. Greenberg, J.P., Mock, S., Bhatia, K., Katz, M., Bruno, G., Sacerdoti, F., Papadopoulos, P., Baldridge, K.K.: Future Generation Computer Systems **21** (2005) 3
2. Sudholt, W., Baldridge, K.K., Abramson, D., Enticott, C., Garic, S., Kondric, C., Nguyen, D.: Future Generation Computer Systems **21** (2005) 27
3. Foster, I., Kesselman, C.: Future Generation Computer Systems **15** (1999) 607
4. Baldridge, K.K., Greenberg, J.P., Elbert, S.T., Mock, S., Papadopoulos, P.: QMView and GAMESS: Integration into the world wide computational grid. IEEE (2002)
5. Nishikawa, T., Nagashima, U., Sekiguchi, S.: ICCS 2003 part III Proceedings Lecture Notes in Computer Science **2659** (2003) 244
6. Kwak, J., Lee, Y.S.: Journal of Theoretical and Computational Chemistry **4** (2005) 289
7. Lesyng, B., Bala, P., Erwin, D.: Journal of Parallel and Distribured Computing **63** (2003) 590
8. http://www.gridchem.org
9. Almond, J., Snelling, D.: Future Generation Computer Systems **15** (1999) 539

10. http://www.scm.com
11. Allen, G., Davies, K., Goodale, T., Hutanu, A., Kaiser, H., Kielmann, T., Merzky, A., Nieuwpoort, R., Reinefeld, A., Schintke, F., Schutt, T., Seidel, E., Ullmer, B.: The grid application toolkit: Towards generic and easy application programming interfaces for the grid. Proceedings of the IEEE **93** (2005) 534
12. K. Keahey and I. Foster and T. Freeman and X. Zhang and D. Galron: Virtual Workspaces in the Grid. In: Europar 2005, Lisbon, Portugal (2005)
13. K. Keahey and K. Doering and and I. Foster: From Sandbox to Playground: Dynamic Virtual Environments in the Grid. In: 5th International Workshop in Grid Computing (Grid 2004), Pittsburgh, PA, USA (2004)
14. http://www.kjemi.uio.no/software/dalton
15. http://dirac.chem.sdu.dk
16. http://dcwww.camp.dtu.dk/campos/Dacapo
17. Bal, H., Bhoedjang, R., Hofman, R., Jacobs, C., Kielmann, T., Maassen, J., Nieuwpoort, R., Romein, J., Renambot, L., Rh, T., an K. Verstoep, R.V., Baggio, A., Ballintijn, G., Kuz, I., Pierre, G., Steen, M., Tanenbaum, A., Doornbos, G., Germans, D., Spoelder, H., Baerends, E.J., Gisbergen, S., Afsermanesh, H., Albada, D., Belloum, A., Dubbeldam, D., Hendrikse, Z., Hertzberger, B., Hoekstra, A., Iskra, K., Kandhai, D., Koelma, D., Linden, F., Overeinder, B., Sloot, P., Spinnato, P., Epema, D., Gemund, A., Jonker, P., Radulescu, A., Reeuwijk, C., Sips, H., Knijnenburg, P., Lew, M., Sluiter, F., Wolters, L., Blom, H., Laat, C., Steen, A.: ACM SIGOPS Operating Systems Review **34** (2000) 76
18. Seidel, E., Allen, G., Merzky, A., Nabrzyski, J.: Future Generation Computer Systems **18** (2002) 1143
19. McCormack, D.: http://macdevcenter.com/pub/a/mac/2005/08/23/xgrid.html
20. Tatipamula, M., Bos E. J.: IEICE Transactions on Communications E878 **3** (2004) 400
21. Nadal, J.: http://www.developer.com/net/net/article.php/3286851