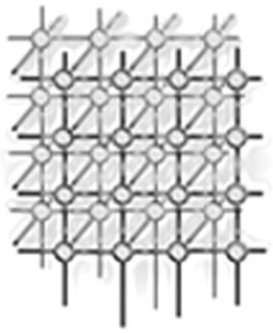
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2004; **16**:385–397 (DOI: 10.1002/cpe.763)



Studying protein folding on the Grid: experiences using CHARMM on NPACI resources under Legion‡

Anand Natrajan[1,*,†], Michael Crowley[2], Nancy Wilkins-Diehr[3], Marty A. Humphrey[1], Anthony D. Fox[1], Andrew S. Grimshaw[1] and Charles L. Brooks III[2]

[1]*Department of Computer Science, University of Virginia, 151 Engineer's Way, Charlottesville, VA 22904, U.S.A.*
[2]*Department of Molecular Biology, The Scripps Research Institute, 10550 North Torrey Pines Road, La Jolla, CA 92037, U.S.A.*
[3]*San Diego Supercomputing Center, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093, U.S.A.*

SUMMARY

One benefit of a computational Grid is the ability to run high-performance applications over distributed resources simply and securely. We demonstrated this benefit with an experiment in which we studied the protein-folding process with the CHARMM molecular simulation package over a Grid managed by Legion, a Grid operating system. High-performance applications can take advantage of Grid resources if the Grid operating system provides both low-level functionality as well as high-level services. We describe the nature of services provided by Legion for high-performance applications. Our experiences indicate that human factors continue to play a crucial role in the configuration of Grid resources, underlying resources can be problematic, Grid services must tolerate underlying problems or inform the user, and high-level services must continue to evolve to meet user requirements. Our experiment not only helped a scientist perform an important study, but also showed the viability of an integrated approach such as Legion's for managing a Grid. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: Legion; computational Grid; CHARMM; protein folding

1. INTRODUCTION

As available computing power increases because of faster processors and faster networking, computational scientists are attempting to solve problems that were considered infeasible until recently.

*Correspondence to: Anand Natrajan, Department of Computer Sciences, University of Virginia, 151 Engineer's Way, Charlottesville, VA 22904, U.S.A.
†E-mail: anand@virginia.edu
‡An earlier version of this paper has been published under the same title in the *Proceedings of the 10th International Symposium on High Performance Distributed Computing (HPDC-10)* [1].

Published online 16 January 2004

Received 6 February 2003
Revised 27 February 2003
Accepted 2 March 2003

Computational Grids are becoming more pervasive platforms for running distributed jobs to solve such problems. A *computational Grid* or a *Grid* is a collection of distributed resources connected by a network. In such an environment, users, such as scientists, can access resources transparently and securely. When a user submits jobs in a Grid, the system runs them on distributed resources and enables the user to access their results during execution and on completion. Developers of Grid infrastructures such as Legion [2] must conduct and present detailed case studies showing how users access Grids. We present one such case study.

Legion is a Grid operating system. It provides standard operating system services—process creation and control, interprocess communication, file system, security and resource management—on a Grid. In other words, Legion abstracts the distributed, heterogeneous and potentially faulty resources of a Grid by presenting users with the illusion of a single virtual machine [3]. In order to achieve this goal, Legion manages complexity in a number of dimensions. For example, it masks the complexity involved in running on machines with different operating systems and architectures, managed by different software systems, owned by different organizations and located at multiple sites. In addition, Legion provides a user with high-level services in the form of tools for specifying what an application requires and accessing available resources.

In our experiment, a computational scientist accessed resources from NSF's National Partnership for Advanced Computational Infrastructure (NPACI) using the Grid infrastructure provided by Legion. The application used was CHARMM (Chemistry at Harvard Molecular Mechanics) [4,5], a popular general simulation package used by molecular biologists to study protein and nucleic acid structure and function. One large problem for which CHARMM is used is the study of the nature of the protein folding process. The scientist desired to study the energy and entropy of many folded and unfolded states of a certain protein, Protein L, to gather information about its behaviour during its folding process and to generate a protein-folding landscape. This study required multiple CHARMM jobs to be run with different initial parameters.

There were two clear goals for this experiment as follows.

1. *Enhance the productivity of the user by solving a large and computationally challenging problem.* By accessing distributed Grid resources, the user condensed the time required for performing his computations from a month (if he used the resources available at his organization) to less than two days.
2. *Demonstrate a match between mechanisms expected by the user and those provided by the Grid infrastructure.* The user had to learn five commands or fewer in order to perform his computations on a variety of resources.

In the process of meeting these goals, we made a number of observations that affect Grid infrastructure developers as well as Grid users. In this paper, we present those observations in the context of the experiment.

Our primary observation was that Grid infrastructures must provide high-level services in addition to low-level functionality. Providing low-level functionality alone is not enough; without high-level services built on top of the underlying infrastructure, a user's productivity can fall tremendously. The novelty of this experiment is not the solving of a large problem (the experiment continues to run at the time of writing), but the ease with which the user accessed Grid resources and the low cognitive burden imposed on the user by the Grid infrastructure, Legion. In this paper, we describe how the user

interacted with a Grid using Legion services, what problems arose with the resources that were part of the Grid and how Legion addressed those problems, and what lessons we learned regarding new functionality that can be provided to users.

In Section 2, we present CHARMM in some detail. The purpose of the discussion is not so much to describe the (fascinating) scientific problem being solved, but to describe the characteristics of the application that make it desirable for running on a Grid. In Section 3, we describe Legion, especially in terms of the features that make it attractive for running high-performance applications. In Section 4, we show how the user interacted with Legion in order to run his jobs on the Grid. In Section 5, we present the results of our experiment. Also, we list the successes of our work, explain problems encountered and identify future directions. We conclude in Section 6.

2. CHARMM

The protein folding process is not well understood and the state-of-the-art methods of studying it are too computationally intensive to be undertaken often. One method is to calculate the free energy surface of the folding process. The calculation is designed to reveal the process by which a small protein (Protein L) folds up into its normal, three-dimensional configuration. The folding process occurs in nature every time a protein molecule is manufactured within a cell. The biophysics of folding must be understood in detail before the information can be used in developing ways of interacting with proteins to cure diseases such as Alzheimer's or cystic fibrosis.

The CHARMM molecular simulation package uses the CHARMM force field to model the energetics, forces and dynamics of biological molecules using the classical method of integrating Newton's equations of motion. Typical systems studied involve protein or nucleic acid molecules of several hundred to several thousand atoms and a bath of solvent, usually water, consisting of many thousands of molecules, for a total of 20 000–150 000 atoms. All chemical bonds and all interactions that do not involve bonds (for example, electrostatics) are used to model the system. These interactions number in the millions to billions. For a typical simulation, hundreds of thousands to millions of timesteps of integration are required and, at each timestep, all interactions are determined. In a parallel run, all forces and all coordinates must be shared among all processors.

CHARMM is computation- as well as communication-intensive. In a single CHARMM job, hundreds of processes may perform computations and communicate with one other. The processes communicate using Message Passing Interface (MPI), a standard for writing parallel programs [6,7]. The parallel efficiency of the computation depends on the number and speed of the processors, and the speed and latency of the interconnect. Since processor speed has increased but interconnect speed has lagged on current-generation high-performance computers, CHARMM's performance degrades rapidly after 32 processors on almost all architectures except the T3E, on which it scales well to 128 processors. Therefore, we chose to run with 16 processors on most architectures, getting better than 95% parallel efficiency throughout the experiment on all high-performance architectures. For a 16-processor run, all processors communicate about 1 MB of data at every timestep in a couple of all-to-all communications, and another 4 MB in each-to-each communications. For a typical 16-processor run on a 375 MHz Power3, approximately three timesteps occur per second. Each job requires a number of input files, some of which are a few megabytes large, and generate a number of output files, some of which are hundreds of megabytes large. Thus, a single job requires powerful computation resources,

fast network capabilities and large amounts of disk space. In our experiment, the user required multiple (up to 400) CHARMM jobs to be run.

We decided to run the CHARMM jobs on a computational Grid because the total amount of computing resources required made it unattractive to run at a single site. Typically, although not necessarily, supercomputing centres such as the San Diego Supercomputing Center (SDSC) use queuing systems to control powerful computation resources connected by fast networks. Since such resources are exactly what CHARMM jobs require, our experiment was conducted on queuing systems. Nothing in CHARMM requires a queuing system; our choice of resources was governed by the coincidence that the kinds of resources that CHARMM requires are usually controlled by queues.

3. LEGION

The Legion project is an architecture for designing and building system services that present users was the illusion of a single virtual machine [3]. This virtual machine provides secure shared objects and shared name spaces. Whereas a conventional operating system provides an abstraction of a single computer, Legion aggregates a large number of diverse computers running different operating systems into a single abstraction. As part of this abstraction, Legion provides mechanisms to couple diverse applications and diverse resources, thus simplifying the task of writing applications in heterogeneous distributed systems. This abstraction supports the performance demands of scientific applications, such as CHARMM. CHARMM runs as a legacy MPI application on queuing systems accessed by Legion. In the following subsections, we discuss some features of queuing systems and Legion's support for legacy applications.

The Grid chosen for running CHARMM was *npacinet*, a nationwide Grid consisting of heterogeneous resources present at multiple sites and administered by different organizations. The majority of the organizations contributing resources to *npacinet* are part of NSF's NPACI thrust. Legion has been managing this Grid continuously for several months during which we have demonstrated Legion features numerous times, conducted tutorials on multiple occasions and supported various academic users running a variety of applications.

3.1. Queuing systems

Queuing systems have been used to schedule jobs on many clusters of nodes [8–12]. When a user submits a job, the queue provides a ticket or job ID or token, which can be used to monitor the job at any later time. The ticket becomes invalid shortly after the job completes. Most queuing systems comply with a POSIX interface requiring three standard tools for running jobs: a submit tool (PBS `qsub`, LSF `bsub`, LoadLeveler `llsubmit`), a status tool (PBS `qstat`, LSF `bjobs`, LoadLeveler `llstatus`) and a cancel tool (PBS `qdel`, LSF `bkill`, LoadLeveler `llcancel`). In addition, some queues provide other tools to check on the aggregate status of the queuing system, e.g., LSF `bqueues` and LoadLeveler `llq`. A queuing system's status tool may report that a job is queued, running or terminated. If the execution of a job is deemed undesirable, the cancel tool can be used to terminate the job. Most queuing systems do not provide tools to access intermediate files or supply additional inputs. A user desiring such functionality must employ shared file systems or other file transfer tools. Queuing systems do not provide any support for checking aggregate progress of large sets of jobs. Users must


```
echo 'Hello, world'
```

Figure 1. Simple application.

```
#!/bin/ksh
#PBS -A anand
#PBS -c n
#PBS -m n
#PBS -N LegionObject
#PBS -r n
#PBS -l nodes=1:ppn=1:walltime=00:10:00
#PBS -p 1
#PBS -o test.o
#PBS -e test.e

echo 'Hello, world'
```

Figure 2. Simple application modified for PBS.

```
#!/bin/ksh
# @ environment = COPY_ALL;MP_EUILIB=us
# @ account_no = met200
# @ class = express
# @ node = 1,1
# @ tasks_per_node = 1
# @ wall_clock_limit = 00:10:00
# @ input = /dev/null
# @ output = test.o
# @ error = test.e
# @ initialdir = /home/uxlegion

echo 'Hello, world'
```

Figure 3. Simple application modified for Maui.

check on the progress of each job individually or construct interfaces to monitor the progress of the entire set of jobs.

Part of the abstraction Legion provides is to hide the differences among queuing systems as well as between queuing and non-queuing systems. A user running over Legion does not have to know the particulars of every system on which a job could run. To appreciate why this abstraction is important, consider running a simple application, such as 'Hello, world', on different systems. We could run on a Unix or Windows system by writing a shell script or batch file such as the one in Figure 1.

However, if we wanted to run on a cluster of nodes controlled by Portable Batch System (PBS), we would have to modify the application to construct a submission script as in Figure 2. If we decided to run on nodes controlled by Maui/LoadLeveler, we would have to construct a submission script as in Figure 3.

Not only are different queuing systems dissimilar, but the same queuing system installed at different sites may be dissimilar in terms of configuration parameters. Moreover, the tools for running special applications, e.g. MPI programs, may be different (`mpirun` versus `pam` versus `poe`). The different submission scripts required to run on different systems restrict a user in two significant ways.

1. The user is forced to learn the particulars of each queuing system, thus increasing his cognitive burden and increasing the time before he can start becoming productive on these systems.
2. When running large numbers of jobs, the user must construct submission scripts for running on each queuing system. The very act of creating a submission script *a priori* forces the user to construct a static schedule for running his jobs. Consequently, he cannot take advantage of dynamic load changes on resources to schedule jobs.

Legion hides differences among queuing systems regarding their submit, status and cancel tools as well as their submission scripts. Also, Legion hides differences regarding the manner in which MPI jobs are run. Moreover, Legion provides tools and mechanisms for accessing intermediate files and viewing the aggregate status of large numbers of jobs. Finally, Legion does not require the user to log on to the various queuing systems to initiate jobs. Single sign-on is one of the most convenient features of a Grid operating system.

3.2. Legacy applications

Legion supports running legacy applications on a Grid. Legacy applications are those that have not been targeted specifically to a Grid or Legion. Legion supports such applications 'as is', i.e. the user neither has to change a single line of code nor re-link the object code to run such an application. All Legion requires are the executables for the application for various architectures. A user who chooses this form of support understands the trade-offs for the convenience of not changing the application at all. One trade-off is that Legion can control very few aspects of the execution of the job after it is initiated. For example, Legion cannot provide restart support for a legacy application if the application itself does not write checkpointing data. However, Legion can and does provide support for starting the job, checking its status as reported by the underlying operating system or queuing system and terminating the job if necessary. In addition, Legion provides the ability to send in or get out intermediate files while the job is running.

4. RUNNING CHARMM ON NPACI RESOURCES

The steps the user had to undertake to run CHARMM over Legion are illustrated in Figure 4. All of these steps were performed after the user logged on (in the Unix sense) to a machine on which Legion had been installed. The shaded boxes represent the steps the user performed without Legion's help. Of these, two, 'Creating Jobs' and 'Analysing Results', are specific to the application. The third, 'Creating Executables', could have been performed with Legion's help. The user had to learn one new Legion command for each of the unshaded boxes. Learning four commands is a small price to pay for the ability to run multiple parallel jobs on distributed heterogeneous resources in a secure and fault-tolerant manner.

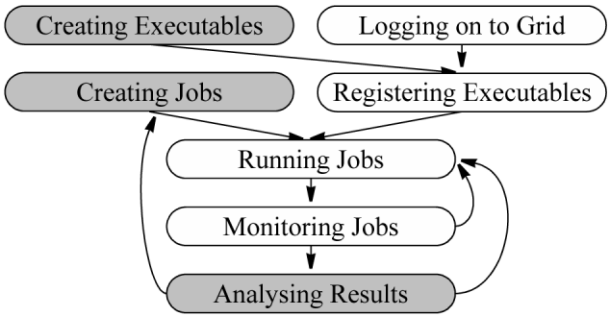


Figure 4. Steps for CHARMM over Legion.

4.1. Creating Executables

In this step, the user created the executables for CHARMM. Recall that the user chose Legion's legacy MPI support for CHARMM. If he desired, he could have used `legion_make`, a tool to compile the source code on machines or architectures of his choosing (in which case, he would have done so after 'Logging on to Grid'). The resulting executables would still be 'legacy code' because Legion would not require changing the source code or linking the object code against Legion libraries. Currently, `legion_make` works for applications with relatively simple and standard make rules, i.e. it works for applications that use standard compilers and have straightforward local dependencies. Since CHARMM is not such an application, the user decided to compile for different architectures without Legion's help.

4.2. Creating Jobs

This step involved creating a set of input files for each job. Clearly, this step is application-specific and requires no help from Legion.

4.3. Logging on to Grid

In order to log on to the *npacinet* Grid, the user ran the command `legion_login`, which required him to enter his Legion ID and password. Once the user logged in, Legion did not require him to log on to any other machine.

4.4. Registering Executables

Registering Executables is the process by which Legion can run a Unix or Windows executable. After an executable is registered with `legion_register_program`, Legion has the information necessary for selecting the appropriate executable to run on any particular machine. Multiple executables of different architectures may be registered with the same Legion object. The benefit is

that a user can request Legion to 'run' the object without having to manage which executable should copied and run on which machine. For example, Legion will ensure that only a Solaris executable is copied and run on a Solaris machine.

4.5. Running Jobs

After registering the executables for every architecture of interest, the user requested Legion to 'run' the object with the command `legion_run`. This command has a number of parameters and options (details are in the Legion man pages accompanying the standard distribution [13]). Parameters for this command include the name of the object and parameters for the job, the names of input and output files for the job, and options such as number of nodes desired, tasks per node desired, duration, etc. Reasonable defaults are chosen for unspecified options. The user may specify a particular machine on which to run or let Legion choose the machine. Likewise, the user may choose to run on any machine of a particular architecture or let Legion make that decision.

The CHARMM user specified the input and output files for each job and the machines on which he desired to run. In addition, the user specified the name of a 'probe file' for monitoring the job (see Section 4.6). The user ran the `legion_run` command as many times as he wanted to initiate jobs. Although the user chose different machines on which to run different jobs (effectively self-scheduling his application dynamically), at no point did he have to write a single submit script, log on to any other machine[§], copy executables and input/output files, or learn a new command for running jobs. The user could have initiated as many jobs as he desired concurrently; in practice, he initiated a few tens of jobs concurrently because: (i) the nature of the jobs imposed sequential dependencies; and (ii) initiating multiple jobs is pointless when the next job is certain to be queued behind previous ones.

4.6. Monitoring Jobs

The user monitored each job in two ways. First, he started a console object for the Unix shell from which he initiated his jobs with one command, `legion_tty`. After the console object was started, output and error messages printed by the user's jobs or the queuing systems on remote machines became visible on the user's shell. Second, the user requested Legion to save a probe for every job. Using the probe and a tool called `legion_probe_run`, the user determined the status of each and every job as well as sent in and got out intermediate files at his leisure. If at any time the user determined that a job was not progressing satisfactorily, he terminated it, corrected any problems and restarted it.

4.7. Analysing Results

The final step involved analysing the results from each job. A basic analysis step involved determining whether each job actually ran to completion. The user made this determination by checking whether

[§]In fact, in the current configuration of Legion on the NPACI machines, the user was not even required to own accounts on the machines. Legion ran his jobs as a generic user on those machines. In the future, the NPACI resources may insist that the user can run on their machines only if he has an account with them as well. Since respecting site autonomy is a critical part of the Legion philosophy, support for the latter mode of operation is under progress.

a certain output file contained specific lines in it. A large part of the subsequent analysis involved retrieving archived files and processing them by running CHARMM again. The subsequent steps were specific to the application and are outside the scope of this discussion.

5. RESULTS

The experiment was conducted successfully over a period of two days. The user logged in to one machine at the University of Virginia on which Legion was installed[¶]. From a single shell on that machine, the user initiated as many jobs as he could, subject to the limitations discussed earlier. Some of the jobs failed, but a large number ran to completion successfully. Consequently, although the user did not manage to complete all of the jobs he desired initially, a significant fraction of the jobs were completed. The experiment showed the viability of running large, high-performance applications on a computational Grid. In the following sections, we discuss how well Legion met the goals mentioned in Section 1.

5.1. Increasing user productivity

A success of this experiment was that the Grid was used to generate results for an actual scientific study. At the time of writing, around 88 of the desired 400 jobs had been completed. We demonstrated that Legion can be used to harness a vast amount of processing power harnessed for scientific users. In the final tally, 1020 processors of different architectures and speeds were utilized for this experiment. The breakdown of these processors is:

- 512 375 MHz IBM Blue Horizon Power3s at San Diego Supercomputing Center (SDSC);
- 128 440 MHz HP PA-8500 at California Institute of Technology (CalTech);
- 24 375 MHz IBM SP3 Power3s at University of Michigan (UMich);
- 32 160 MHz IBM Azure Power2s at University of Texas (UTexas);
- 32 533 MHz DEC Alpha EV56s at University of Virginia (UVa);
- 260 300 MHz-nodes Cray T3E at SDSC;
- 32 400 MHz Sun HPC 10000s at SDSC.

In the future, we intend adding the following resources:

- 88 300 MHz Cray T3Es at UTexas;
- 32 400 MHz dual-CPU Intel Pentium IIs at UVa.

We estimate that if the user had used the resources available at his organization alone (128 SGI Origins), it would have taken one month to complete what was complete in less than two days on the Grid. The number of jobs run on each resource is shown in Figure 5. The vast majority of the jobs ran on the Blue Horizon at SDSC because that machine was by far the most powerful machine in the mix of available machines. Some of the machines did not contribute significantly to the results because of run-time problems (see Section 5.3).

[¶]Legion was not installed on the user's machines at The Scripps Research Institute (TSRI) because of site-specific firewall restrictions.

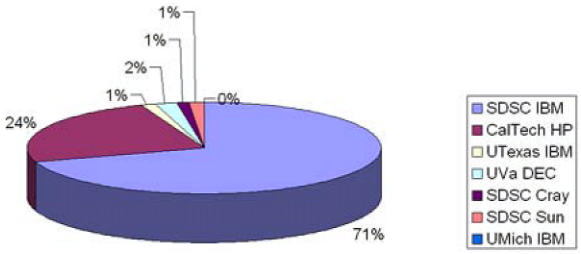


Figure 5. Breakup of CHARMM jobs completed.

5.2. Simplifying Grid access

Legion's ease of use could be measured in what the user had to do as well as what he did *not* have to do to run his jobs. As described in Section 4, the user had to learn a mere four or five commands to run on the Grid. The small number of commands is comparable to the number the user would have to learn for *each* queuing system had he not chosen Legion. During the experiment, the user did not have to log on to any of the queuing systems. He logged on to one machine at UVa on which Legion was installed. From a single shell on that machine, he initiated multiple jobs. Legion made the heterogeneous NPACI resources available to the user without his having to know the details of how to run on each resource. The heterogeneity of the resources extended in a number of dimensions:

- six organizations (UVa, TSRI, SDSC, UTexas, UMich, CalTech);
- six queue types (Maui, LoadLeveler, LSF, PBS, NQS);
- up to ten queuing systems;
- up to six architectures (IBM AIX, HP HPUX, Sun Solaris, DEC Linux, Intel Linux, Cray Unicos).

5.3. Identifying and eliminating problems

A number of run-time problems caused fewer total jobs to complete. Minor organizational problems aside, the problems we encountered fell into two categories: network slowdowns and site failures. The Legion run-time system suffered no problems during the experiment, although a number of potential extensions were identified (see Section 5.4). Also, although the CHARMM user used the Grid heavily, the remaining users on the same Grid were unaware of the experiment. While the experiment progressed, other Legion users continued to run their usual jobs on the Grid.

5.3.1. Network slowdown

During the experiment, we experienced slowdowns in the network connections between UVa and SDSC. From around noon through about 3 P.M. US EST, medium-sized to large packets were transmitted from one site to the other with great difficulty. Preliminary investigation showed that

packets of size equal to or greater than 8800 bytes were lost entirely. Packets in the size range 8000–8800 bytes suffered over 90% loss rates. The loss rates for packets of size less than 8000 bytes were lower but still significant. The implication for Legion was that some messages between objects had to be retransmitted a number of times to ensure that they were received correctly. Consequently, for the CHARMM user, monitoring jobs became a slow process. At one point, inquiring about the status of a job took nearly a minute to complete. Ordinarily, this process is almost instantaneous. Since the user could not monitor jobs quickly enough to start new ones, throughput was reduced.

5.3.2. Site failures

Some of the NPACI sites experienced unforeseen failures. For example, at UMich, Legion encountered NFS failures. Since the ability to access permanent storage is important to Legion as well as CHARMM, the NFS failures reduced the throughput of CHARMM jobs. On the Blue Horizon machine at SDSC, the queuing system, Maui/LoadLeveler, had to be restarted a number of times because it became overloaded. During the time the queuing system was down, currently running jobs continued to run. However, the queuing system could not inform anyone about the status about those jobs. Since 'no information' is similar to what the queuing system reports when a job has been complete for a while, Legion assumed the jobs were complete and informed the CHARMM user accordingly. This erroneous reporting led the user to believe that it was safe to access the output files from the job. However, on analysis of these jobs, the user discovered that the output files were only partially complete. At UMich, the purge policy in place removed CHARMM files as well as persistent state required by Legion objects. Without their persistent state, Legion objects can behave erroneously. Likewise, without the appropriate input files CHARMM cannot run as intended.

5.4. Extending Legion

The experiences of the CHARMM user enabled us to identify potential extensions to Legion. These extensions would enhance Legion's usability by building on low-level functionality already present.

5.4.1. Graceful error handling

Legion has been designed to mask many kinds of failures from end-users. While this strategy usually benefits the user, sometimes it is important for the Grid infrastructure *not* to mask failures from the user. For example, the network failures discussed earlier were masked from the user who saw only gracefully degraded performance. However, Legion also masked most site failures from the user, which often conveyed the mistaken impression that Legion itself had failed. Consequently, we are reviewing all aspects of error handling and propagation in Legion.

5.4.2. Support for archiving

Although Legion permits users to specify input and output files at any time during the execution of a job, archival support is almost non-existent. In particular, there is no way for a user to specify that some files are meant to be stored on some kind of long-term storage after the job is complete. Instead,

the Legion file solutions are that after a job is complete, the files are either copied out to the user's local directories, or to Legion's own distributed shared file system, or deleted. None of these solutions is satisfactory for jobs that generate large amounts of data. The user's local directories or the individual components of the distributed file system may not have space to store large amounts of data. Moreover, the user may not want to copy files out the moment the job is done. Instead, scientific users generating large amounts of data, such as the CHARMM user, are likely to want to archive the data generated by their jobs on some long-term storage and access the data at their leisure. Since Legion developers did not anticipate such a need, currently, archiving has to be done by users themselves as part of their jobs.

5.4.3. Support for parameter-space studies

We are making it possible for users to run parameter-space studies of parallel codes with a single command. The CHARMM user had to issue a fresh `legion_run` command for every job. Legion provides another tool, `legion_run_multi`, which enables multiple jobs to be started with one command. This tool works well if every job is a sequential program; we are looking to extend it to parallel programs.

5.4.4. Web interfaces

We are developing a Web portal that scientists can use to run CHARMM jobs. Currently, we have a Legion Web interface for the entire Grid [14]. We intend adding a CHARMM-specific component to this interface.

5.5. Observations

1. *High-level services:* the CHARMM experiment reassured us that a Grid infrastructure must provide low-level functionality *and* high-level services. We consider it a significant advantage that using Legion, the user accessed heterogeneous resources controlled by multiple organizations with four or five commands and achieved order-of-magnitude speedup as compared with running at just one site.
2. *Human Factors:* the humans involved in the experiment were critical to the experiment's success. Three people were involved intimately with the continuous progress of the runs: the user, a Legion liaison and an NPACI liaison. The Legion liaison was present in case problems arose with Legion itself during the execution. Since Legion itself suffered no run-time problems, this person used Legion tools to identify site-specific problems as they arose. The NPACI liaison coordinated on-site efforts to keep the experiment running. Finally, administrators at individual sites ensured that problems were resolved as soon as possible by correcting misconfigurations, restarting services, increasing quotas, etc. Although this collaboration was rewarding, in the future the involvement of all parties except the user must be eliminated.
3. *Site Services:* the number of site failures that were identified was astonishingly high. Normally, users never expect services such as queues and operating systems to fail. Likewise, users rarely consider network failures when running their applications. However, running large numbers of high-performance jobs can stress-test every component of a Grid. We discovered previously ignored limits on the number of jobs queues can manage, queue-imposed job duration limits,

credential expirations with file systems, purge policies, process table limits, quota exhaustions and numerous other problems, each of which could make a site unusable for continued running.

6. CONCLUSION

We demonstrated that Legion is a suitable environment for running large, high-performance jobs, such as CHARMM, on a Grid. Legion provides a suite of tools for a Grid that are similar to what traditional operating systems provide for a single system. Using these tools, users can start, monitor and terminate jobs on remote machines in a straightforward manner. The CHARMM runs on heterogeneous machines controlled by different organizations and showed that Legion is able to mask unwanted detail from end-users, thus permitting them to focus on completing their work. Although we encountered a number of problems during the run, it is encouraging to note that none of the problems are unsolvable; solutions for each of them are forthcoming from the lessons we have learnt. These solutions can only improve and simplify a user's access to a computational Grid.

ACKNOWLEDGEMENTS

We thank Dave Carver at UTexas; Sharon Brunett and Mark Bartlett at CalTech; Victor Hazlewood, Larry Diegel and Kenneth Yoshimoto at SDSC; Tom Hacker and Rod Mach at UMich; and Katherine Holcomb and Norm Beekwilder at UVa for providing the CHARMM user with accounts, providing Legion accounts and allocations, and resolving problems if and when they arose at their respective sites.

REFERENCES

1. Natrajan A, Crowley M, Wilkins-Diehr N, Humphrey MA, Fox AD, Grimshaw AS, Brooks CL III. Studying protein folding on the Grid: Experiences using CHARMM on NPACI resources under Legion. *Proceedings of the 10th International Symposium on High Performance Distributed Computers*, August 2001.
2. Grimshaw AS, Wulf WA. The Legion Vision of a worldwide virtual computer. *Communications of the ACM* 1997; **40**(1):39–45.
3. Grimshaw AS, Ferrari AJ, Lindahl G, Holcomb K, Metasystems. *Communications of the ACM* 1998; **41**(11):257–270.
4. Brooks BR, Bruccoleri RE, Olafson BD, States DJ, Swaminathan S, Karplus M. CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. *Journal of Computational Chemistry* 1983; **4**:187–217.
5. MacKerell AD Jr, Brooks BR, Brooks CL III, Nilsson L, Roux B, Won Y, Karplus M. CHARMM: The energy function and its parameterization with an overview of the program. *The Encyclopedia of Computational Chemistry*, vol. 1, Schleger PVR *et al.* (eds.). Wiley: New York, 1998; 271–277.
6. Hempel R, Walker DW. The emergence of the MPI message passing standard for parallel computing. *Computer Standards and Interfaces* 1999; **7**:51–62.
7. Snir M, Otto S, Huss-Lederman S, Walker DW, Dongarra J. *MPI: The Complete Reference*. MIT Press: Cambridge, MA, 1998.
8. Bayucan A, Henderson RL, Lesiak C, Mann N, Proett T, Tweten D. Portable batch system: External reference specification. *Technical Report*, MRJ Technology Solutions, November 1999.
9. International Business Machines Corporation. *IBM LoadLeveler: User's Guide*, September 1993.
10. Kingsbury BA, The network queueing system (NQS). *Technical Report*, Sterling Software, 1992.
11. Zhou S. LSF: Load sharing in large-scale heterogeneous distributed systems. *Proceedings of Workshop on Cluster Computing*, December 1992.
12. Zhou S, Wang J, Zheng X, Delisle P. Utopia: A load sharing facility for large, heterogeneous distributed computer systems. *Software: Practice and Experience* 1993; **23**(2).
13. *The Legion Manuals*, vol. 1.7. University of Virginia, October 2000.
14. Natrajan A, Nguyen-Tuong A, Humphrey MA, Herrick M, Clarke BP, Grimshaw AS. The Legion Grid Portal. *Concurrency and Computation: Practice and Experience* (*Special Issue on Grid Computing Environments*) 2002; **14**:1365–1394.

