An Integrated User Environment for Scientific Cluster Computing

Niclas Andersson, Peter Kjellström
National Supercomputer Centre
Linköping University, Sweden
{nican,cap}@nsc.liu.se

Abstract

We provide an integrated environment to the users of clusters for scientific computation. The environment simplifies the overall usage of the system and makes it easier for new users to learn and start using the system. It is an attempt to integrate the various pieces of software that comprise the user environment of an open-source based Linux cluster without introducing limiting constraints. We target this environment towards users who consume the system's volatile resources (CPU time and network bandwidth) to do research in physics, chemistry, and other application areas. It does not require any additional code in the application as long as the used communication paradigm is supported on the system. Instead, researchers with own code find it easy to build and run their program.

This work also extends into the realms of system administration, where tools and the configuration of the system make it feasible to provide more informative services to users.

This environment is currently in use on two clusters which are used extensively for research in physics and chemistry.

1 Introduction

On computer systems used for scientific computation, a plethora of tools and libraries is often installed to support researchers in their work. On Beowulf systems, clusters built using primarily commodity of-the-shelf (COTS) parts [14], the system software mainly consists of open-source software. It is often comprised of many disparate pieces of software not very well attuned to each other. Consequently, there is a lack of integration in the software environment presented to the user; tools have little or no knowledge of each other, parameters need to be mentioned multiple times, lengthy paths need to be specified, and so forth. For instance, the program `mpirun`, commonly used to start parallel MPI (Message Passing Interface [10]) applications, is seldom aware of resource managers. Therefore, users sometime find it difficult to start applications on the right set of processors. In this case, the lack of integration between `mpirun` and the resource manager not only makes it more difficult to use. It may also affect other users and hence the overall utilization of the system. Our goal is to simplify the user enviroment, improve integration between tools, and provide a stable system for production in a supercomputer center.

To improve the integration of system software and simplify the user evironment, we make modifications and additions to installed software. The NSC[1] cluster environment (NCE) we provide does not restrict the choice of which pieces of software to install and use. it will always be possible to extend and incorporate more tools. Neither do we hide the original interface of the tools, presenting a completely new layer to the user. This would require substantial work, packaging and repackaging for each new version of the underlying software. Also, such additional layer would not be readily adopted in the open-source community. Instead, the tools and configurations we use is made aware of each other to simplify the overall use. This is achieved by making small modifications and additions to existing packages.

A short review of related work is given in section 2, Section 3 and 4 describes the production environment and programming environment which shape the landscape for the main tasks a user is faced with; to build and run applications. In section 5 we explain the details of the current implementation.

2 Related Work

In this section we will present an brief look at related work done in the field of cluster toolkits and environ-

[1] NATIONAL SUPERCOMPUTER CENTRE IN SWEDEN

ments. The projects fall more or less into two different categories, cluster installation and management, and Linux environment support.

There are several toolkits available to aid in the installation and maintenance of Beowulf systems. These toolkits have slightly different purposes and covers slightly different aspects of running a HPC cluster.

Open Cluster Group's OSCAR [11] toolkit can be described as package of cluster related software bundled with an GUI/wizard style installation tool. Advantages with the OSCAR concept includes single point access to updated software and a user-friendly installation process. Configuration is stored in a MySQL database on the front-end machine and node installation is handled by Linux Utility fo cluster Installation (LUI).

NPACI Rocks [12] is a cluster distribution that strives to be easy to install and maintain. It is designed to support heterogeneity in software as well as in hardware thus offering a very flexible solution. To simplify node management, Rocks employs the idea of stateless nodes. The default action is to reinstall the node using Red Hat's kickstart installer. Rocks includes a very well designed and well maintained base of tools to take care of installation and every-day maintenance. Like OSCAR, Rocks uses a MySQL database to store cluster configuration data. Rocks is developed by the Distributed Development Group at SDSC and the Millennium Group at UC Berkeley.

Both OSCAR and NPACI Rocks are examples of flexible cluster solutions with complete Linux installations on every node of the cluster. They focus on system installation and maintenance, whereas NCE focus on the provided user environement.

The Scyld Beowulf [13] offers an architectural remake of the Beowulf cluster concept presenting a more or less single process space across the cluster. This approach offers many new possibilities but also raises some compatibility issues with software (eg. PVM) which has to be tailored for the new environement. The single process space gives the user a much more manageable environment. Scyld Beowulf main contribution is an integrated operating system across separate, clustered computers.

Cactus [1], an open source problem solving environment (PSE), offers a framework of utilities to computation requirements. It provides extensive services to the user such as parallel I/O, data distribution, and checkpointing. To obtain and enjoy these services the application has to be interfaced with the framework and use the provided toolkit. This is easily done when developing new software from scratch, whereas legacy code can be more difficult to accomodate. There are several more PSEs available with different levels of granularity and different modularity schemes. A single, complete programming model for parallel applications is not on the horizon.

Modules [4] (and its various offsprings) is a UNIX package to bundle commands for setting and modifying user environment variables such as `PATH` and `MANPATH` in a dynamic fashion. It is designed to facilitate having many, possibly conflicting, software packages and versions on a single system and makes it feasible to select which to use and switch between them. For users that never have experienced Modules, it is a of course new concept. However, it is easy to learn and simplifies the handling of common user environment variables.

3 Production Environment

A *production environment* is composed of the various pieces of software that are used to execute and monitor applications in a reliable and controlled manner. For small clusters and clusters used by single research groups, the out-of-the-box operating system installation may be sufficient. For larger systems, systems installed in remote computing centers, and clusters on which many groups compete for the available resources, additional software for resource management and scheduling as well as monitoring and accounting is needed.

3.1 Resource Management

A job submitted for execution on a computer system must have the possibility to allocate the resources it claims it need to run to completion. It is the resource management (RM) system's responsibility that these requirements are fulfilled.

In a desktop computer, the operating system's process scheduler is typically the only present resource manager. This is often adequate as long as there are a limited number of users. In servers for scientific computing with many users present, there is a need for more manageable and tunable RMs. Such RMs often takes the shape of a daemon implemententing a batch queue system although its data structures and priority systems has little resemblance with queues.

There are many resource managers available, both commercial and open sorce. A few examples are: Load Sharing Facility (LSF), Network Queueing Environment (NQE), Distributed Queueing System (DQS), Portable Batch Scheduler (PBS) and Condor. Although many of these RMs are delivered with a builtin job scheduler, their abilities to efficiently schedule parallel jobs on a cluster with many nodes are highly deficient. PBS (which also exists in an open source version; OpenPBS) provides the necessary set of limits and features. Additionally, PBS has an open interface to external schedulers. This makes it possible to use PBS together with

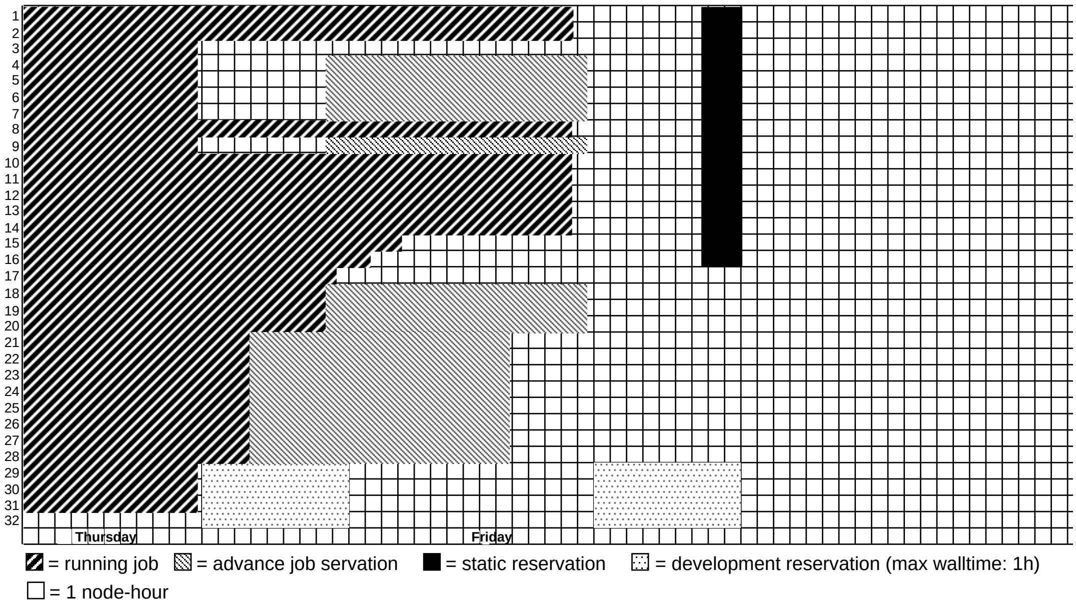

Figure 1: A WWW-page showing running jobs and submitted but not running jobs (advance reservations) on the Beowulf Ingvar in a lucid time-node diagram. One square represents one node-hour.

for example the Maui job scheduler. (More on this in the next section.)

3.2 Job Scheduling

Fair scheduling of parallel jobs on a cluster is a difficult task. There are many schedulers for batch queueing systems available making half-hearted attempts which result in underutilized systems and unfair distribution of resources among users. One common technique of these schedulers is to sort the jobs in a strict priority order and start the job with the highest priority when the requested resources becomes available. This is an adequate technique as long as the required width (number of hosts) for most of the jobs are identical (eg. 1). However, often they yield a low utilization when jobs are parametrized on the number of nodes (hosts). To raise the utilization, lower priority jobs are often allowed to overtake and start when higher priority jobs are delayed because of insufficient resources. The result is possible starvation of highly parallel jobs since they constantly may be overtaken by smaller jobs. This is a common problem on highly utilized systems. It can be eliminated by introducing an additional rule, a "starve protection" rule based on maximum allowed queueing time, and raise priority or, more likely, force jobs to start when they have reached this limit. However, this causes other scheduling problems, eg. higher indifference to job priorities.

One effort to provide "fairness" in the use of computing resources is fair-share scheduling [7]. This approach bases the current priority of a job upon the use history of its owner. Since CPU-cycles are volatile and thus can not be saved for later use, this technique is questionable [8].

The Maui Scheduler [6] is, a generally better approach: If there is a current lack of resources to start the next job in line, make a reservation of the requested resources in the future! The notable difference from most other scheduling efforts is the presence of these advance reservations in a future schedule, taking the dimension of time into account. The schedule which is recreated repeatedly by Maui resembles a meeting calendar where the meetings are jobs and where the scheduler makes constant efforts to fill every gap but still paying respect to the job priorities.

We have found that the Maui scheduler provides a surprisingly good scheduling strategy for a Beowulf cluster. The ability to make advance reservations, facilitates complex scheduling decisions and solves many of the problems other schedulers have. Moreover, Maui's recurrent dynamic scheduling makes it easy to change the priority of a job if necessary.

3.3 Accounting

While the job scheduler can provide a fair use on a day to day basis, the long term use of computing resources has to be managed according to other principles such as invested money and granted resources. For this purpose, accounting information is collected and used to steer the long-term scheduling policy. The raw accounting data can be collected from several different sources. The

```
eric% projinfo
Project/User Used[h] Granted[h] Priority
----------------------------------------
2001218       807.33    1500.00   Normal
| bill          0.00
| donald      784.23
| eric         23.12
```

Figure 2: `projinfo` displays granted and used resource for all the projects the user is member of.

most basic technique is process accounting collected directly from the operating system kernel. It is the most exact method where the CPU time of every process is accounted for separately. However, difficulties may arise in correlating these data with sessions and jobs, especially in clusters where many kernels are involved. Furthermore, in cluster where computing nodes are regarded as dedicated resources, the process accounting generates superfluous information for the accounting record.

Another source of accounting information is the resource manager system. Although this system has no records of interactive time, time used without the resource managers knowledge (eg. compilations and small tests), there are additional information such as size and length of jobs and requested amount of resources. The gathered accounting information can be used not only to debit the consumed resources correctly but also to limit the maximum use allowed for a single user or project. Additionally, measurements for tuning and statistics such as the level of utilization of the system, the average job size, and the average wait time may also be collected.

In NCE, the command `projinfo` displays the amount of granted and used CPU-hours on the cluster. For each project the user is member of, it gives the project's granted resources and the resources used by each project member (see figure 2)

At NSC, when a project exceeds its granted resources the priority of its member's jobs are lowered for the rest of the accounting period. The change in priority is forwarded into the job scheduler which only schedules low priority jobs when all jobs of normal priority have received thier requested resources. In the absence of jobs with normal priority, the resource may still be utilized by projects with overdrafted accounts. In the Maui scheduler this is implemented by assigning different levels of quality of service (QoS) to different users.

3.4 Monitoring

To get a general and quick overview of running and submitted jobs on a cluster, a dynamically updated WWW-page is provided (see figure 1). It presents the information in a time-node diagram, a format that is easy to grasp. It has quickly become a useful tool for both users and system administrators.

During execution, both the job owner and the system managers should have the possibility to monitor the progress of the jobs. By examine the job output, the job owner can check the job's progress and decide if the job should be aborted prematurely and free the resources. In NCE, the command `pbspeek` can be used to display the stout or stderr of a running job. The system manager is concerned with the efficient usage of the system and with the proper functioning of its components, both hardware and software. For this purpose there are plenty of different tools available. These tools are not included here since they are primarily targeting the system administrator and not the users of the cluster.

4 Programming Environment

To build applications, a supportive programming environment is necessary. On a cluster for scientific computations the necessary components includes:

- Compilers producing efficient code,
- Libraries for high bandwidth and low latency communication, and
- Highly optimized scientific libraries.

Unless a separate system is avilable for development, efficient tools for profiling and debugging are also desired on the production cluster.

In the quest for maximal performance, it is essential that alternatives to the main compilers, communication software, and libraries can be installed and easily used. However, to provide a wide variety of tools to users and still keep them simple and intuitive to use is not an easy task.

4.1 Compilers

Most scientific high performance applications are written in Fortran. Fortran 90 is the standard of today and since the freely available g77 in the GNU Compiler Collection (GCC) only implements Fortran 77, a Fortran 90 compiler has to be purchased. Portland Group's compiler suite (PGI Workstation) is selected for its good performance and acceptance of several language flavors. Beside the Fortran 90 compiler (`pgf90`) PGI Workstation also contains a Fortran 77 compiler (`pgf77`), a C compiler (`pgcc`), a debugger (`pgdbx`) and a profiler (`pgprof`). Since their default names differs from the Linux native GCC tools (`g77`, `cc`, `c++`, `gdb`, and `gprof`) there are no difficulties to make all of them available to users. The user can easily access them after installation by extending the search path to the installed binaries.

In both PGI Workstation and GCC it is possible to add local options to the tools without writing additional drivers or wrappers. This possibility is utilized to simplify the choice of communication libraries as described in the next section.

4.2 Communication Libraries

Next after compilers, the most important system software on a Beowulf are the communication libraries. They provide the application program interface (API) for fast communication among the collaborating processors within an application. Both MPI and PVM implement the message passing paradigm and are accustomed communication APIs in scientific computing. The most widely used implementations include MPICH [3], LAM [2], and PVM [5]. For the fast SCI network installed on Ingvar (see section 5), the proprietary MPI implementation ScaMPI from SCALI.

Instead of choosing one package for each of MPI and PVM, there are reasons for supporting them all:

- The proprietary ScaMPI is the only library available for the fast SCI network interfaces.
- Applications which are compiled elsewhere can easily use any of the communication libraries and launch methods.
- There are differences in performance and portability between the different MPI packages.
- Functionality differs between implementations.

This availability of interchangeable packages presents a challenge to the integrated easy-to-use approach.

Even if the application programming interface (API) is almost identical in ScaMPI, LAM, and MPICH, implementation details such as the names and number of libraries needed during linkage differ. Compiler wrappers provided by some packages (eg. `mpicc` shipped with MPICH) eliminates the path problem from the include and library paths. Instead, it becomes a combinatorial problem when multiple compilers for one implementation language is available. For two different compiler suites and three different MPI implementations, six different compiler wrappers is needed just for the C language. Additional problems arise when other APIs, not related to MPI or PVM, also require the use of compiler wrappers. To circumvent this, NCE provides a more orthogonal approach by defining local compiler options. The difference from using the normal include and library paths is that we can encapsulate lenghty paths and files needed in a short, easy to remember, word or abbreviation. For instance, to compile `pingpong.c` with PGI's C-compiler `pgcc` and MPICH, we can use the full paths to include and library directories:

```
pgcc -I/usr/local/mpich-pgi/include \
  -o pingpong pingpong.c \
  -L/usr/local/mpich-pgi/lib -lmpich.a
```

or the the designated compiler wrapper, `mpicc`:

```
/usr/local/mpich-pgi/bin/mpicc \
  -o pingpong pingpong.c
```

or the NCE option:

```
pgcc -Nmpich -o pingpong pingpong.c
```

Furthermore, there are a wide variety of ways to launch an appliacation:

1. Even though MPICH, LAM, and ScaMPI all provide and use a program named `mpirun`, these can not be used interchangeably; MPICH's `mpirun` can not be used to launch an application linked with neighter LAM nor ScaMPI and vice versa.
2. Applications linked with PVM is always started on one single processor.
3. Both LAM and PVM requires that necessary daemons, `lamd` or `pvmd`, has been initialized beforehand on the allocated nodes.

To simplify and to present a unified launch method to users, NCE provides `mpprun` which can launch MPICH, LAM, ScaMPI as well as PVM applications. It uses the designated method for each communication library. To interactively start the previously built `pingpong` on two nodes, we simply give the command

```
mpprun -np 2 ./pingpong
```

`mpprun` will request two nodes from the resource manager. It will also discover that it is linked with MPICH and issue the necessary commands to run the application with MPICH's `mpirun`.

4.3 Scientific Libraries

Optimized libraries for calculations such as BLAS and Lapack are essential components in the provided programming environment. The highly optimized ATLAS library [15], which implements BLAS level 3 is easily built and installed. Just as in the case of communication libraries, an extra compiler option is provided for these, most common, libraries.

5 Implementation Details

We initially implemented and installed these environment enhancements on the class II Beowulf system *Ingvar*Ingvar is now one of the resources available to researchers throughout Sweden via the Swedish National Allocation Committee (SNAC). It is used extensively for research in mainly physics and chemistry.

The system consists of a front-end used for login, compilations, job submissions, and file services and 32 compute nodes. All 33 computers are connected to a switch using Fast Ethernet. Furthermore, the 32 compute nodes are equipped with SCI-based interfaces built by Dolphin. The SCI networks performance is far better than the Fast Ethernet (88.2 MByte/s node-to-node bandwidth, 4.7 μs latency [measured]) and is suitable for communication intensive applications.

The distribution we use is Red Hat Linux 6.2. The installation of the nodes is done by configuring one node and use VA Systemimager to clone the disk to the other 31 compute nodes. For day-to-day administration, we keep a centralized copy of some of the files in `/etc` which we synchronize using `rsync` when necessary.

Crucial to every easy-to-use approch is a good choice of default values. For instance, without any action taken from the user, environment variables such as `PATH` and `MANPATH` should have values that allow the user to access binaries and man-pages for the most common software packages installed on the cluster such as compilers, resource manager, and monitoring commands. However, when there are several software packages available to choose from and none of them is an obvious alternative, it is necessary to require that the user make an active choice. To this end, none of the installed `mpicc`-wrappers are found by using the default `PATH`.

5.1 Compiler Options

The local compiler options NCE provides `-Nscampi`, `-Nlam`, `-Nmpich`, `-Npvm`, `-Nblas`, `-Nlapack` can be implemented either by writing a *site wide* compiler wrapper[2] or, as we have managed to do so far in NCE, by modifying configuration files for the existing compiler drivers. Both PGI and GCC have configuration files with which to specify the connection between the options given to the compiler driver and the options the driver should add to the separate compiler stages. Eg. the rule

```
*lib:
%{Nmpich:-/usr/local/mpich-gcc/lib/
  libmpich.a}
```

[2]in contrast to the *package wide* compiler wrappers often enclosed in different packages

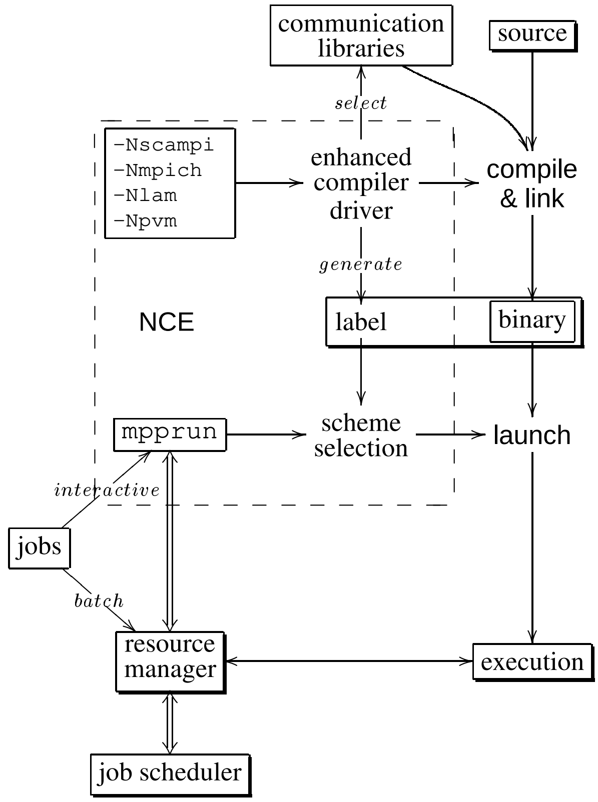

Figure 3: Besides selecting the appropiate communication library the local compiler options also labels the binary. This label is used to set the `mpprun` default launching scheme.

specifies that the MPICH library is passed to the linker when you supply the option `-Nmpich` to `gcc`. All of the above mentioned local options are possible to implement in both GCC and PGI by adding rules to configuration files.

Furthermore, an extra object file with a single unique symbol is used to label the binary. This symbol facilitates an unambiguous identification of used libraries and thus provides a channel of information from compile time to execution time (see figure 3).

5.2 `mpprun`

NCE's `mpprun` is inspired by the Cray T3E command by the same name. While T3E uses the global resource manager (GRM) daemon within the kernel to allocate the requested resource, NCE communicates with PBS for resource negotiations.

`mpprun` first checks if it is invoked within a PBS job script. If so, the number of nodes PBS has allocated for the job becomes the number of instances of the application to launch. `mpprun` further determines which launching scheme to use by examining the unique symbol described in section 5.1. In case this symbol is absent (application is linked elsewhere or the local compiler options were not used), heuristic methods are instead used to guess the appropriate lauching method. Since there is a possibility that the wrong method is selected, it may be overridden with options supplied by the user[3]

If `mpprun` is invoked interactively from a shell, an interactive job[4] is submitted to PBS using `expect` [9] The I/O connection between application and terminal is retained as for any other interactive command. Consequently, starting parallel applications become as easy as starting sequential applications or commands.

We keep the configuration for each installation of the communication package in its original form to simplify uppgrades and future installations. This also allows users to continue to use the original schemes of library paths or compiler wrappers (eg. `mpicc`).

5.3 PBS and Maui

PBS has daemons (`pbs_mom`) continuously running on every node. `pbs_mom` is responsible for launching, monitoring and killing application instances on one node. PBS' central resource manager, `pbs_server`, contacts each mom frequently to gather status information and issue commands in accordance with the scheduler.

The use of the Maui job scheduler, reduces the needed configuration in PBS significantly. Features in Maui such as policies, quality-of-service, and standing reservations supersede the resource controlling features in PBS. Only one single unlimited default "queue" remains necessary.

Crucial to the production environment is that all resources that a job has allocated are released and made available to future computations. To enforce this, we supply PBS with an epilogue script which kills running user processes and removes all files in the scratch directories local on the nodes. `pbs_mom` is instructed to execute this script after each job.

6 Future Work

The changes and additions we have applied are not easily encapsulated in a automatically installable package. It requires knowledge of the combination of the used softwares to correctly apply the tools and scripts that "glue" them together. However, we will pursue a method to readily apply these changes and relase them under an open source license.

[3] To achieve consistency in the user environement, the same options are used here as for the compiler: `-Nscampi`, `-Nmpich`, `-Nlam`, `-Npvm`.

[4] a PBS job with option '`-I`'

In the current programming environment only the basic functionality is provided so far. We plant to extend the framework to include support for debugging, profiling, and performance measurements.

On classic (non-Scyld) Beowulf clusters where nodes can be accessed with `rsh` or `ssh`, a user can easily bypass the resource manager. To strengthen the system integrity, a more restricted access policy can be enforced by modifying the user access to nodes on a job to job basis. The Pluggable Authentication Modules (PAM) can easily be used to dynamically allow or deny specific users.

The Modules package mentioned in section 2 can be used to further enhance NCE. When accustomed to Modules, it is a very efficient tool when configuring and handling the settings in a user environement.

7 Results

The result of this work is *not* another distribution or administration package to make installations and administration of Beowulfs easier. On the contrary, NCE could be successfully employed together with such packages.

The environment alleviates the users of some of the difficulties usually experienced when starting to use a Beowulf system. We provide a user environment which is more consistent and easier to use then most users currently expect from a Beowulf system. The concealment of lengthy tool options reduces the demand for extensive documentation on how to use and combine the numerous existing options. It also eases the task of reconfiguring the system eg. the placement of libraries. Furthermore, options are mentioned in fewer places, eliminating errors and increasing the number of successfully launched jobs.

We have shown that, with small changes, the integration between tools can in an open-source environment be significantly improved. Details, such as consistency in options (section 4.2 and 5.2) and an easy-to-grasp WWW-page (section 3.4) can have a large impact on new users. The script generating the WWW-page is now included as contributed work in the Maui scheduler distribution.

NCE is currently in use on two Beowulf systems. Both of them are available as nation-wide computing resources for academic research. With the increasing number of Beowulfs, NCE makes it possible to present an integrated, uniform environment to users on several clusters, even if the selection and the location of installed packages are not identical. We currently offer the users two compiler suites, three different MPI implementations, PVM, and scientific libraries. Still, the need for supportive help to new users is remarkably low.

Acknowledgement

We want to thank Anders Ynnerman for his efforts, without which this work would have been impossible. Anders initialized NSC's involvement in commodity clusters. We also want to thank Zach Brown for his valuable help in proofreading and correcting this paper. Finally we thank the anonymous reviewers for their comments and suggestions.

References

[1] Gabrielle Allen, Werner Benger, Tom Goodale, Hans-Christian Hege, Gerd Lanfermann, André Merzky, Thomas Radke, Edward Seidel, and John Shalf. The cactus code: A problem solving environment for the grid. In *Proceedings of the Ninth High Performance Distributed Computing Conference*, pages 253–260, Pittsburg, PA, 2000. IEEE Computer Society.

[2] G. Burns, R. Daoud, and J. Vaigl. LAM: An open cluster environment for MPI. In *Proceedings of Supercomputing Symposium '94*, pages 379–386. University of Toronto, 1994.

[3] N. E. Doss, W. Gropp, E. Lusk, and A. Skjellum. An initial implementation of MPI. Technical Report MCS-P393-1193, Mathematicas and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, 1993.

[4] John F. Furlani. Modules: Providing a flexible user environment. In *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)*, pages 141–152, San Diego, CA, September 1991.

[5] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994. The book is available electronically at `ftp://www.netlib.org/pvm3/book/pvm-books.ps`.

[6] David Jackson. Maui scheduler on Linux. Workshop notes and presentations, USENIX Annual Technical Conference, June 1999.

[7] J. Kay and P. Lauder. A fair share scheduler. *Communication of the ACM*, 31(1):44–55, January 1988.

[8] Richard Klamann. Opportunity scheduling: An unfair CPU scheduler for UNICOS. In *Proceeding of CUG Meeting*. Cray User Group, 1997.

[9] Don Libes. expect: Scripts for controlling interactive processes. *Computing Systems*, 4(2), November 1991.

[10] Message Pasing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3):157–416, 1994.

[11] OSCAR: Open source cluster application resources. `http://www.csm.ornl.gov/oscar/`.

[12] Philip M. Papadoupoulos, Manson J. Katz, and Greg Bruno. NPACI Rocks: Tools and techniques for easily deploying manageble linux clusters. Submitted to Cluster '01, The Third IEEE International Conference on Cluster, October 2001.

[13] Scyld Beowulf clustering for high performance computing. `http://www.scyld.com/`.

[14] Thomas L. Sterling, John Salmon, Donald J. Becker, and Daniel F. Savarese. *How To Build a Beowulf: A guide to the Implementation And Application of PC Clusters*. Scientific and Engineering Computation Series. MIT Press, 1999.

[15] R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimization of software and the ATLAS project. `http://www.netlib.org/atlas/`.