Cluster Computing 6, 267–278, 2003
© 2003 Kluwer Academic Publishers. Manufactured in The Netherlands.

The Rutgers Computational Grid: A Distributed Linux PC Cluster

B. CHERNYAVSKY *
Department Mechanical and Aerospace Engineering, Rutgers – The State University of New Jersey, Piscataway, NJ, USA

E. GALLICCHIO
Department of Chemistry, Rutgers – The State University of New Jersey, Piscataway, NJ, USA

D. KNIGHT
Department of Mechanical and Aerospace Engineering, Rutgers – The State University of New Jersey, Piscataway, NJ, USA

R. LEVY
Department of Chemistry, Rutgers – The State University of New Jersey, Piscataway, NJ, USA

A. PAGE
Telecommunication Division, Rutgers – The State University of New Jersey, Piscataway, NJ, USA

Abstract. The Rutgers Computational Grid (RCG) project is aimed at providing high throughput performance to Rutgers university faculty and students. The RCG employs dual processor PCs, with Pentium II and III processors, as computational nodes, running the Linux RedHat operating system. The Load Sharing Facility (LSF) scheduling system from Platform Computing is used for job control and monitoring. The nodes are grouped into subclusters physically located in several departments and controlled by a single master node through LSF. The hardware and software used in RCG are described. Utilization and performance issues, including parallel performance, are discussed based on the experience of the first two years of RCG operation.

Keywords: grid computing, scheduling system, parallel computing

1. Introduction

One of the new and important directions in modern computing is the *computational grid* [7], usually described as a collection of clusters, or 'super-cluster', consisting of inexpensive, PC-based machines and managed by a scheduling system. Foster and Kesselman [10] describe five primary classes of grid applications: distributed supercomputing, in which grid resources are used to solve very large problems; high-throughput computing, in which grid resources are used to solve large numbers of small tasks; on-demand, in which grids are used to meet peak needs for computational resources; collaborative, in which grids are used to connect people; and data-intensive, which focuses on coupling of distributed data resources. Dongarra [8] provides a number of examples of grids in development or operation, implementing all or some of this applications. The Rutgers Computational Grid (RCG), described in this paper, is optimized for an academic environment. Such an environment is characterized by a diverse mix of computational requirement for different academic units, irregular periods of peak activity followed by the periods of relatively limited usage and a mixture of a parallel and scalar jobs (see [13] for another example of computational grid for academic environment). Berkley Network Of Workstations (NOW) [3] present another example of distributed computational grid, although composed mostly from Sun workstations rather than PCs. Therefore, the RCG is primarily utilized for the second and third of these purposes: to provide a high-throughput computing platform and to provide additional resources for peak-performance periods. The clusters comprising the RCG are physically distributed with individual groups of machines located in remote locations (and owned by different academic units) and primarily used to support local users while also providing additional CPU resources for peak periods of computational grid utilization when jobs which exceed the limits of a local group of machines can be dispatched to others which have available CPU resources. We refer to a group of machines physically located in one place and belonging to one owner as a *subcluster*. The size of jobs submitted to a computational grid are usually larger than would be feasible to run on a personal workstation, but usually do not exceed capability of the local subcluster. Typically, grids such as the RCG utilize a batch scheduling system, controlled by a single master. Allan [1] provides a survey of different batch scheduling systems available for this purpose. We choose LSF batch scheduling system [17] for this purpose as a flexible and reliable commercially supported system.

This paper describes the configuration of a high throughput performance computational grid (the Rutgers Computational

* Corresponding author.
E-mail: boris.chernyavsky@ae.gatech.edu

Grid – RCG) established at the Rutgers University and discusses the experience and lessons learned during the first two years of its operation from January 2000 to January 2002. Utilization statistics, including experience with parallel codes, development of the batch queue system and possible directions of future work are discussed.

2. System description

2.1. Hardware

The Rutgers Computational Grid (RCG) is comprised of 58 dual processor PC computational nodes. The processors are a mixture of 400 MHz Pentium IIs (Deschutes) and 450, 500 (Katmai), 550, 850 and 1000 MHz Pentium IIIs (Coppermine). The history of the RCG development is summarized in the table 1. Most machines have 512 MB RAM, except for one with 1 GB RAM and four others with 256 MB RAM. The Engineering nodes were upgraded to 1 GB RAM and Psychology nodes to 512 MB RAM in October 2001. Most of the nodes have 7 GB IDE hard drives which are used only for system files. Several machines each have an additional 20 GB hard drive which serves as user directories. The nodes are distributed in five *subclusters* located in different academic units (figure 1 illustrates RCG configuration on September 2001). Three of the subclusters (Chemistry, Engineering and Physics) are located on the New Brunswick campus. The Psychology subcluster is located on the Newark campus (approximately 23 miles to the north) and the Computer Science subcluster on the Camden campus (approximately 58 miles to the south). The three campuses are connected by a 155 Mbps ATM link. All of the machines within the subclusters are connected through a 100 Mbps Fast Ethernet network, while connection between subclusters is provided through a 10 Mbps network.

2.2. Software

The operating system on the RCG machines is RedHat Linux Version 6,1, 6.2 and 7.2, Load Sharing Facility (LSF) [17], a scheduling system developed by Platform Computing, provides scheduling and batch service for the cluster. The RCG utilizes LSF version 3.2 Standard Edition which includes Basic LSF and LSF Batch. It should be noted that neither LSF

Table 1
RCG hardware addition history.

Date	Subcluster	Number and type of processors added to RCG
Jan 10	Engineering	16 Pentium II 400 MHz
Apr 13	Chemistry	10 Pentium II 400 MHz
Apr 20	Physics	10 Pentium II 400 MHz
Apr 28	Psychology	6 Pentium III 500 MHz 1 Pentium II 400 MHz
Jun 5	Engineering	12 Pentium III 550 MHz
Aug 2	Physics	16 Pentium III 550 MHz
Aug 7	Chemistry	12 Pentium III 550 MHz
Aug 8	Physics	12 Pentium III 550 MHz
Nov 8	Computer Science	8 Pentium III 450 MHz
Apr 1	Engineering	12 Pentium III 850 MHz (replacement)
Aug 1	Engineering	4 Pentium III 1000 MHz 4 Pentium III 550 MHz (replacement)
Sep 25	Psychology	2 Pentium III 500 MHz (replacement) 8 Pentium III 500 MHz

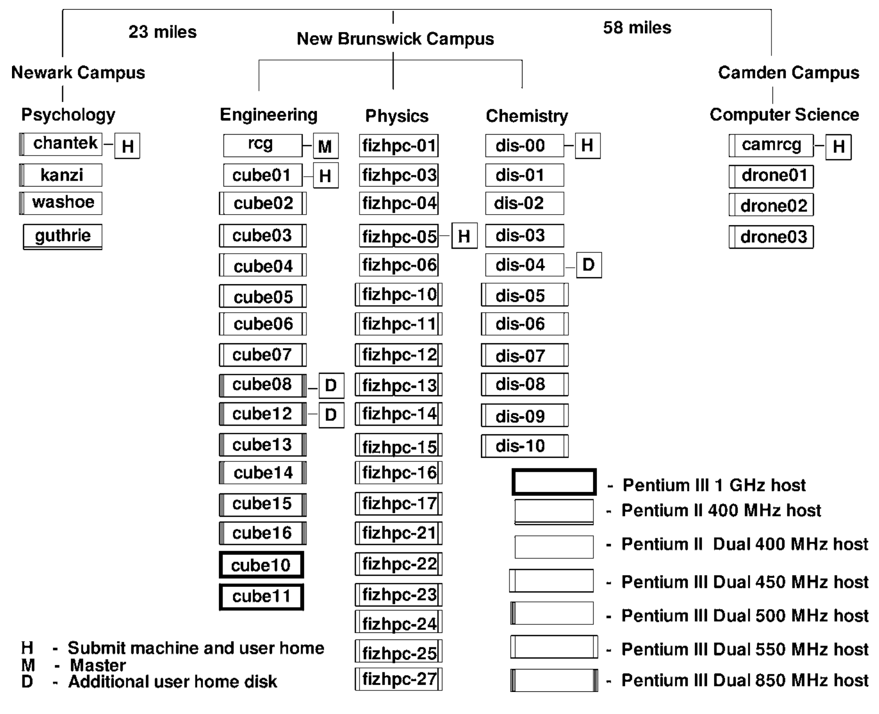

Figure 1. RCG cluster configuration.

Multicluster nor LSF Parallel software was acquired. All machines, irrespective to which subcluster they belong, have the same master, so the term "subcluster" should not be confused with "subcluster" in LSF literature, which refers to fully independent subclusters managed through LSF Multicluster. The Mpich 1.2.0 release [18] of MPI is used for parallel jobs on machines belonging to a single subcluster. No MPI jobs across the subclusters are permitted, since the efficiency of message passing decreases dramatically when machines from different subclusters are involved due to the high network traffic between subclusters.

2.3. System configuration

All of the machines are controlled by a single master, which also serves as a license server and provides disk space for the common software used by all or most of the nodes (e.g., compilers, MPI parallel software and specific tools which individual academic units would like to be available on the RCG or on the specific subcluster machines). Each subcluster has its own login machine which is the only machine available for users from that academic unit. This machine also mounts the users' home directories on a dedicated hard drive. Additional hard drives containing extended home space are available for the Engineering and Chemistry users. To ensure user space uniformity, users' home directories are exported to all participating machines. Thus, a job is always executed in the user's home directory, regardless of which particular node it is executed on, unless the user specifically requests otherwise (for example a job requiring significant I/O may be given access to a local scratch disk on the execution machine, which can be achieved by a simple script). User accounts are maintained through Network Information Service (NIS), which insures uniform availability of each machine for a job submitted by any user.

The parallel codes are implemented with the MPI protocol, using `mpich 1.2.0` realization. `Mpich` supports both C and FORTRAN compilers installed on the RCG. The RCG provides users with PGI and Gnu C and FORTRAN compilers. Individual academic units may also install additional software for their users.

Since one of the primary aims of the RCG project is high throughput computing using a batch scheduling system, users are not permitted to run interactive jobs on the RCG. This does not present a significant inconvenience for users, since most users have access to Linux PC workstations, nearly identical to individual nodes of the RCG, and therefore able to develop their codes and process the results of computations outside of the grid without having to convert data file formats and executables. Users can request that they be automatically notified by the system through email when their job either completes or exits.

The distribution and execution of submitted jobs is controlled by a system of batch queues. Modifying or creating additional queues allows a high degree of flexibility and allows the LSF administrator control over resource allocation, job priorities and limits, etc. Queues can be configured to provide selected users (or groups of users) with different priorities, to optimize matching job requirements and available resources (such as RAM, availability of certain software on specific group of nodes or ensuring that all nodes executing MPI job would be taken from the same subcluster) and to maintain a uniform level of resource utilization across the grid. Detailed description of the configuration can be found on the RCG website [19].

3. Performance

One of the important goals of the RCG development was to provide users with a low cost platform for parallel computing. To evaluate the performance of the grid in a multiprocessor mode, a representative Computational Fluid Dynamics (CFD) code [16] performing Large Eddy Simulation (LES) of isotropic turbulence decay was used as a benchmark. The code uses the `mpich` distribution of MPI. The code is parallelized by dividing the computational domain into independently processed subdomains for each processor, with halo cells layer exchange employed to maintain domain unity. Figure 2 illustrates the speedup demonstrated for a test problem on 64^3 grid. It shows that the scalability remains good for both Pentium II and III based nodes, with predictably lower speedup for faster nodes, due to the lower calculation to communication time ratio. Figure 3 illustrates the results of the *scaled speedup* benchmark for parallel jobs with different number of processors. For this benchmark, the grid was modified to keep a constant size number of cells for each processor. The benchmark was performed using two C compilers, PGI C and Gnu C, both with and without optimization options. It is seen that the plot is practically horizontal, indicating good scalability. It also shows the absence of significant network related degradation of performance inside a single subcluster, justifying use of inexpensive Fast Ethernet network.

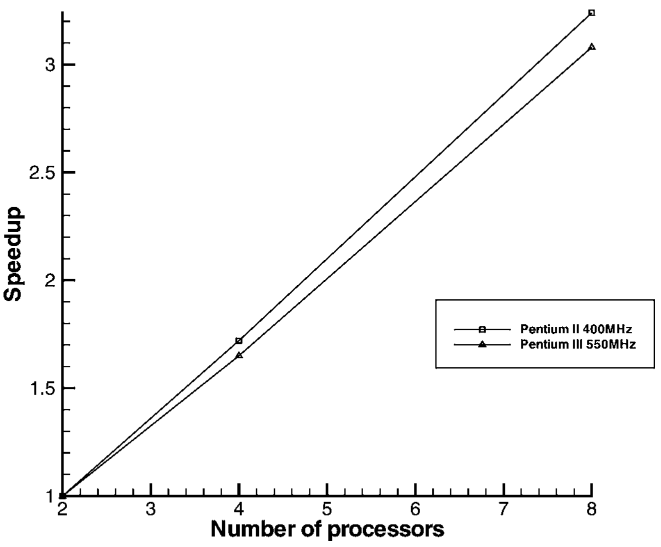

Figure 2. Non-scaled speedup, LES, 64^3 grid.

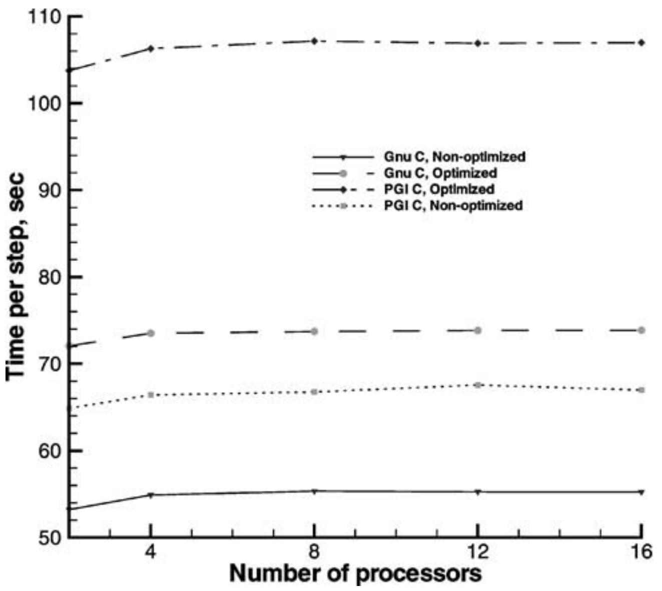

Figure 3. Scaled speedup, LES, 64^3 grid.

4. Analyses of experience

4.1. Queue system

The individual subclusters, comprising computational grid, are tied together through the batch scheduling system. Its function is to control the distribution of submitted jobs between and within individual subclusters according to specific resource requirements of a particular jobs, available resources and established policies to achieve a maximum efficiency of the grid as a whole. The choice of the scheduler and job distribution algorithm is therefore of particular importance for achieving high performance of the computational grid. The choice of an optimal scheduler strategy is a matter of an active discussion among grid computing community, with a number of both custom designed and commercially available batch scheduler software packages being discussed (see, for example [4–6,11]).

The efficient scheduler should support a flexible hierarchy satisfying established priority and quota policies in multi-user environment as well as providing high performance of the grid as a whole. The problem of allocating a mutually acceptable share of computational resources to all users is particularly important for the computational grid, since its components are often owned by different entities with individual user groups and specific requirements. Therefore, it is often necessary to create a hierarchy of user groups and individual users with individual priority, available resource and resource share allocations. A number of different algorithms realizing such hierarchy, including fair-share [12,15] and proportional-share [5] scheduling were proposed. This user resource allocation algorithm should also be realized in a way compatible with the requirement of providing high performance of the whole grid by efficient load balancing. One of the primary load balancing issue being discussed is the realization of the optimal scheduling in heterogeneous environment characteristic for the computational grid architecture. The examples are discussed in [4,6]. Another difficult problem is the optimization of the mix of both scalar and parallel jobs, characteristic for many computational grid. The issue is discussed in details in [2,4,9]. In most of discussed cases, however, the efficient scheduling optimization would require some advance knowledge of application properties, which is not always available, particularly when, as in our case, computational grid is used as a research tool and most of the applications are user-developed. Optimization of a mix of parallel and scalar jobs represent a particular problem if it is combined with grid heterogeneity [6].

A number of shareware and commercially supported batch scheduler software packages are available on the market, such as Condor, Cluster Controller, LSF, PBS, etc. (see, for example [1] for survey of some of the available options). For the RCG we choose the Load Sharing Facility (LSF) [17] designed by Platform Computing due to its flexibility, easy of use and availability of customer support. The LSF control job distribution and resource allocation through a set of batch queues, which can be configured with necessary features, such as priority setting, fair-share or user resource quotas, etc. Each queue can be configured both to be accessible only by the specific group of users, and to distribute submitted jobs to a specific group of hosts, creating additional flexibility in matching users with computational resources according to a specific policy. A large number of configurable options in the queues provides a high degree of flexibility and allows the LSF administrator control over resource allocation, job priorities and limits, etc., by creating a set of queues.

The RCG was created to provide a high-throughput computing platform in the academic environment, characterized by a very diverse range of application and significant variation of load with time. Each of the five participating academic units has an unique set of application executed on the RCG, resulting in different computation resources requirements. Academic units also have different number of users, peak utilization period frequency and duration, and user access policy. It is also important to notice that the primary function of the RCG is that of the research and development tool, and the majority of the application executed on the RCG are user developed. This make it often difficult or impossible to accurately predict in advance the requirements of computation resources, or their variation with time. This diversity and unpredictability made the development of mutually acceptable resource allocation policy particularly difficult. Therefore, we adopt an empirical approach, observing the realistic pattern of load evolution, and using it later for development of a queue system tailored for our environment.

During the pilot phase (January–October 2000) of the RCG operation it was configured with a queue set deliberately restricted to a small number of the simple queues. The aims of this initial queue system were to support the primary grid function of job distribution between the subclusters during peak periods and allow accumulation of the experience in order to formulate the requirements for a more advanced queue system for the production phase. The basic set of queues was configured based on a proprietary principle, i.e., designed to attempt execution of the job submitted by a member of a given academic unit on the node belonging to that academic unit. Each of the academic units has their own queue and only members of that academic unit have an access to corresponding queues. All the queues have the same priority. This arrangement provides for the most jobs being executed on 'proprietary' subcluster, therefore justifying their purchase for corresponding academic units, while allowing to use free resources on other subclusters during peak utilization periods, when one (or several) of the academic units have requirement for more nodes than it is available in the 'proprietary' subcluster. No fairshare or fixed limitation on the number of jobs submitted by a given user and/or academic unit was implemented and the jobs are executed by First Come-First Served principle. One of the aims of this simple arrangement was explicitly to observe the "natural" load distribution which may serve as a basis for developing the production phase queue. In addition, this system allowed a simple way of estimating the relative amount of computation done by different academic units (since all but excessive jobs are executed on their own subclusters), and to a significant extent leaves control of the subcluster in the hands of their owners. Only when the amount of the submitted jobs exceeds the capability of the local subcluster do the jobs start to be dispatched to the available processors on the other subclusters, demonstrating the most important feature of the computational grid: an efficient way to accommodate 'peak loads' by redistribution of excess load to remote subclusters.

Three additional queue types were implemented at the start of the RCG operations. One of them was requested by one of the academic units for the purpose of code development and testing and was configured to give jobs submitted to it higher priority but limiting runtime for any job to fifteen minutes in order to avoid a negative influence on other jobs. Another queue which was suggested and implemented at the beginning of the pilot phase was a traditional for batch scheduling systems night queue, allowing users to submit a large number of jobs which would be run during the 'quiet hours'. Another special queue(s) were created for submission of the MPI job which will be discussed separately.

The practical experience with this simple set of queues proves highly successful. Practically all serial jobs were submitted to the 'standard' proprietary queues. A small job queue, created explicitly for code testing and debugging, was left almost unused, despite the initial request for its creation. The primary explanation is that the RCG nodes are off-the-shelf PC nodes, nearly identical to the personal workstations and all code development and processing of the results can be done there without necessity to recompile executables, rendering small jobs queue redundant. Similarly, experience of the RCG operation show that there were little advantage of using the night queue in the RCG environment. Analyses of LSF log files shows that many jobs were actually launched in the interval 10 pm–6 am (actually this time produced more than 20% of all submitted jobs), traditionally considered as a "low load time". Since most of the jobs run at least a full day, the load on the system varies insignificantly with the time of the day, rendering a night queue unnecessary. On the other hand, practical experience indicates the necessity of creating another queue. Most of the Chemistry users are using the commercial Gaussian software in conjunction with their code. It was observed by several users that when two of such jobs were running on the same node, the runtime significantly increased, most probably due to concurrent use of the same software by two processors on the same node, which would often lead to simultaneous disk access requests from different processes, slowing down job execution. As a corrective measure a special queue was created which limited submission of jobs from that queue to one per node, effectively removing the reason of the slow down.[1] Another special queue was created for users running jobs with the use of `Matlab` software, specifying a set of nodes on which this software was configured. The issue of grid heterogeneity, increasing with each new upgrade, was addressed by setting CPU values in resource configuration files, which ensured that the job would be always dispatched on the fastest available CPU (albeit only within a subcluster, due to a 'proprietary' principle of queues organization).

Despite original intention to implement a fairshare, or fixed quotas, on the amount of runtime available for individual users and/or academic units based on condition of jobs waiting in the queue after the end of the pilot phase, this simple queue set was retained throughout most of the first year of the production phase. While it was noted that a very small group of users utilized a seemingly disproportionate amount of time exceeding the time of all other users combined, the limited number of "friendly users", utilizing the RCG during the pilot phase, and fast expansion of the RCG itself resulted in a limited applicability of obtained load pattern information to a full production phase operations and do not indicate the necessity of the immediate alteration of queue system. In this situation the inherent difficulty of formulating a mutually acceptable policy for the five academic units with different numbers of users, runtime and work load requirements led to delaying the decision until it become truly necessary and sufficient information become available to design the superior queue set.

Sufficient amount data on utilization pattern was observed in the course of the first year of production phase to justify rethinking of the queue system. It was confirmed, that despite significantly larger number of users, a small number of users are still capable to utilize unproportional share of computation resources, prompting requests for introduction of fairshare and/or job submission limit. While the full utilization of all RCG nodes was observed only during relatively short periods of time, the overall utilization level was consistently high (see utilization section for details). In this condition a group of long running jobs were able to 'displace' jobs with a shorter runtime to the nodes with slower CPU, which prompt suggestion of separation of jobs of significantly different length into different queues. Also, the grid was observed to be saturated by simultaneous submission of very large number of relatively small jobs, which led to establishing of the upper limit for the total number of jobs submitted, both by individual users and academic units. These experience resulted in the development of new queues set commissioned in fall 2001.

The new queue set is still based on 'proprietary' principle, with each academic unit now having three basic queue, for the short (less then one day), medium (between one and five days) and long (between five and twenty five days) jobs with the limitation on the number of jobs in each queue which can be submitted simultaneously which was derived from observed load pattern. Within each queue, user job priority is determined by the fairshare, with priority dropping proportionally to number of jobs user submit during a certain time interval.

However, the academic environment may present situations, such as when a group of users need to conclude their calculations before a fixed deadline, when using of fairshare may be impractical. In order to resolve such situations a special queue is currently under consideration, which would allow a group of designated users to temporary use a higher priority for execution of their jobs, including displacing and suspending the lower priority jobs already in process of execution by the RCG.

4.2. Parallel jobs

With increasing popularity of the parallel computing, one of the major goals of the RCG project is to provide a user-friendly environment capable of supporting parallel jobs with a moderate number of processors. Due to the concept of the RCG as a cost-efficient, high-throughput system, it was not intended or optimized to handle massively parallel tasks. However, as was previously shown, it demonstrates a high efficiency for the small-to-medium size parallel jobs. The size of the parallel jobs on the RCG is limited both by technical and policy considerations. Due to the limited speed of inter-subcluster communication, it is obviously impractical to run a parallel job on the nodes belonging to different subclusters, which limit a theoretical maximum size of the job to the capacity of the largest subcluster. Special queues were created for the Engineering and the Physics academic units for submission of parallel jobs to ensure that all requested processors would be chosen from the same subcluster. Since the preferred policy at this stage is to try to accommodate local users on their own subcluster, separate MPI specific queues were created for different academic units, limiting execution hosts to ones belonging to local subcluster. In the future, however, it might be preferable to implement a single parallel queue, which would submit MPI jobs to any of the subclusters which have sufficient number of available processors, irrespective to the code ownership.

Another important issue for the parallel jobs is the influence of processor inhomogeneity. The acquisition of the RCG nodes over an extended period of time resulted in a different processor being utilized on individual nodes. Our tests demonstrates that the benchmark MPI code, when run on a collection of processors even with close CPU speeds (e.g., 400 and 550 MHz) suffers a significant penalty. Given that the benchmarks on 550 MHz nodes demonstrate 1.52 times higher speed than on 400 MHz nodes we observe only a 55–70% processor usage on the faster nodes. The unused CPU time is spent waiting for the slower nodes to complete calculations. Such losses are obviously increasing with the increase in the processor speed disparity. Users had reported an experience, when a parallel job, being sent on the different types of processor, require longer time to complete than a similar sized scalar job submitted to the quicker node. To avoid such waste of computational resources, the parallel queues were further refined to include only homogeneous processors, farther restricting the acceptable job size.

This restriction, however, was reinforced by the policy consideration of the RCG as a high throughput system, intended to provide resources for a large number of users with a mix of a scalar and parallel jobs. Obviously, massively parallel jobs denying the CPU resources for the other users for an extended period of time can lead to unproportional allocation of available resources to the single user. On the other hand, massively parallel jobs were often waiting for extended periods of time in the queue until the requested number of processors became available simultaneously on a given subcluster. Therefore, restricting the size of the parallel jobs to the number of heterogeneous nodes in a given subcluster solves both the problem of parallel execution inefficiency and conflict between parallel and scalar job requests. With reduced maximum parallel job sizes it becomes practical to introduce processor reservation. If the scheduling system is unable to find sufficient number of appropriate processors at a moment of job submission it would try to reserve every additional processor as they become available until the required amount can be reached. By setting higher priority to the parallel jobs it is possible to ensure that this reservation would have higher priority than scalar jobs submitted in the general queue. The reservation can be withdrawn, if the requested number of processor is not attained after a specified maximum time. With the modification of basic queue system in the fall 2001, this system was modified to include back-fill option. This option allows to back-fill scalar jobs to be executed on the reserved processors if its expected execution time is less than time which would be required to attain the necessary number of processors for parallel jobs. This option, while obviously attractive, was not implemented to date, as it would require users to specify explicitly code execution time; since the RCG is used primarily as a research instrument, however, it is usually difficult or impossible to predict code running time with any accuracy, making this option seldom used. With the new basic queue set, separated by expected maximum length, this options becomes practical. While it is often not possible to determine the exact time the job would require to complete, it is usually possible to estimate upper limit of this time to separate job into short, medium and long runtime categories. That would allow to back-fill any job submitted into the short runtime category.

Experience with parallel operations indicate one potential vulnerability of the MPI codes, namely high sensitivity to memory chips errors. A number of MPI jobs failed due to apparently faulty memory chips. These failures led to the

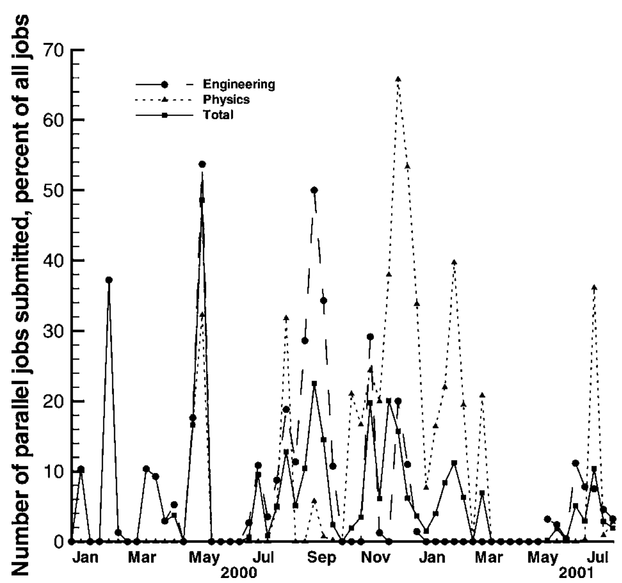

Figure 4. Percentage of MPI jobs, 10 day increments.

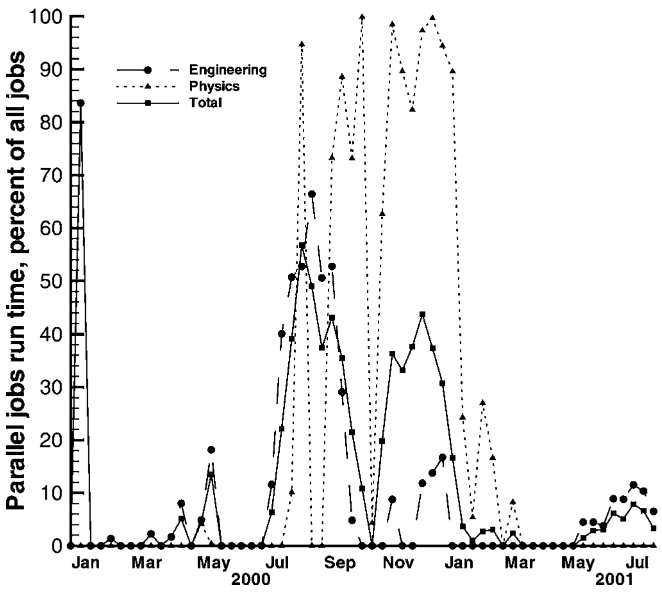

Figure 5. Percentage of time used by MPI jobs, 10 day increments.

Table 2
Distribution of waiting time.

Time waiting in a queue	Percent from total	
	All jobs	Parallel jobs
< 2 min	82.4	87.7
2 min–1 h	4.4	4.5
1 h–3 h	3.7	2.3
3 h–8 h	3.4	2.8
> 8 h	6.1	2.7

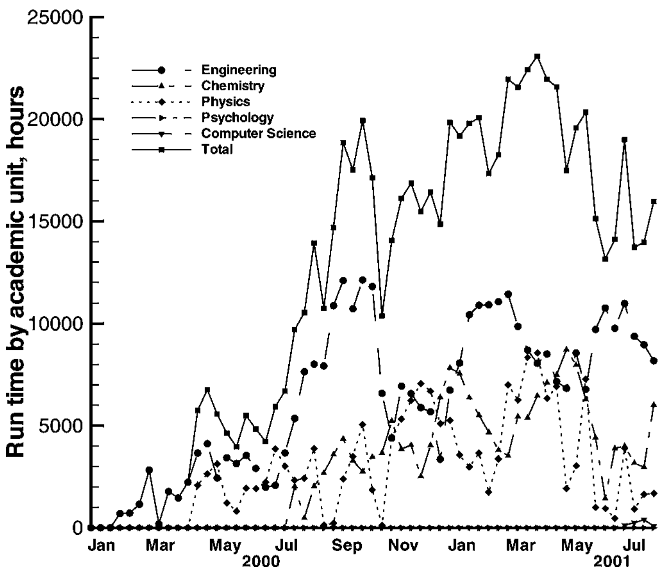

Figure 6. Runtime hours for individual academic units, 10 day increments.

undesirable situation where an MPI code failed on some of the nodes, while the others nodes continued to wait for the signal from them until user manually killed the remaining processes. This resulted in a significant loss of runtime. This indicates the necessity to implement a pre-execution script which would be started on all of the nodes chosen for execution of the MPI code and would control its status, sending a termination signal to all processes used by the MPI code in case of failure on any particular processor.

Despite the fact that only five users actively used MPI in their codes (figure 11), parallel jobs represent a significant share of the RCG workload. At certain times near the end of the pilot phase, MPI jobs represented nearly 30% of all submitted jobs and used almost 50% of total runtime, although a value around 20% of CPU time is more common, making parallel jobs an important component of RCG operations (see figures 4 and 5). It should also be noted, that average queue waiting time for submitted job is acceptably low (see table 2). 87.7% of the parallel job start execution less than 2 minutes after submission, i.e., they receive requested number of processors immediately. Only 2.7% of all parallel job wait in the queue more than 8 hours.

4.3. Utilization

During the first two years of operation, thirty one users from four academic units were actively[2] utilizing the RCG. Figures 6 and 7 summarize evolution of the RCG load in terms of wall clock run time used and number of jobs submitted in ten day increments. Figure 6 shows continuous growth of the run time hours during first twenty months of operation. It should be noted that during this time the processors were upgraded and the same number of runtime hours in September 2001 corresponds to significantly large CPU resource than in January 2000.

The figures 6, 7 and 8 shows total run time, number of jobs and utilization percentage by users from each academic unit regardless of where the job executed on the RCG. Although the utilization level is subject to sharp changes due to such factors as the time of the year and addition of new nodes to the grid, the average utilization is approximately 75% and during two month in the spring 2001 exceed 90%. Despite this high average utilization level (and short periods of grid satu-

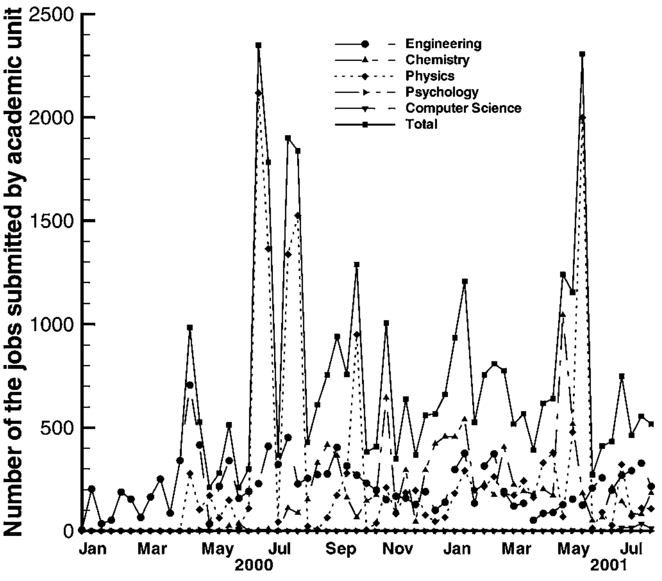

Figure 7. Jobs submission for individual academic units, 10 day increments.

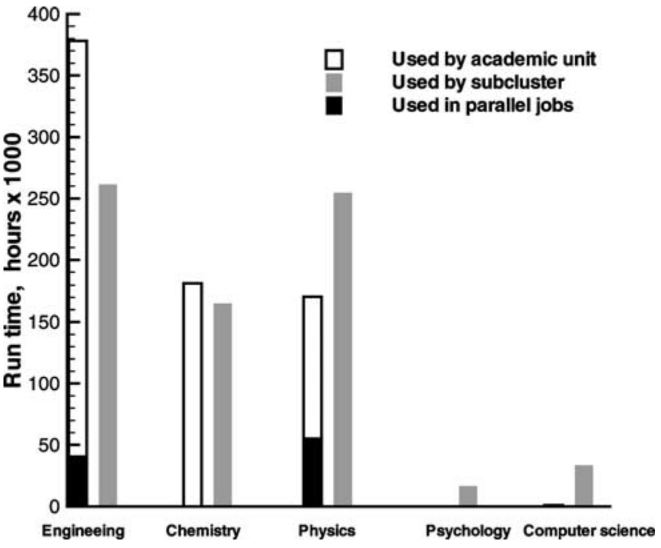

Figure 9. Comparison of the RCG utilization by academic units vs. subclusters (January 2000–September 2001).

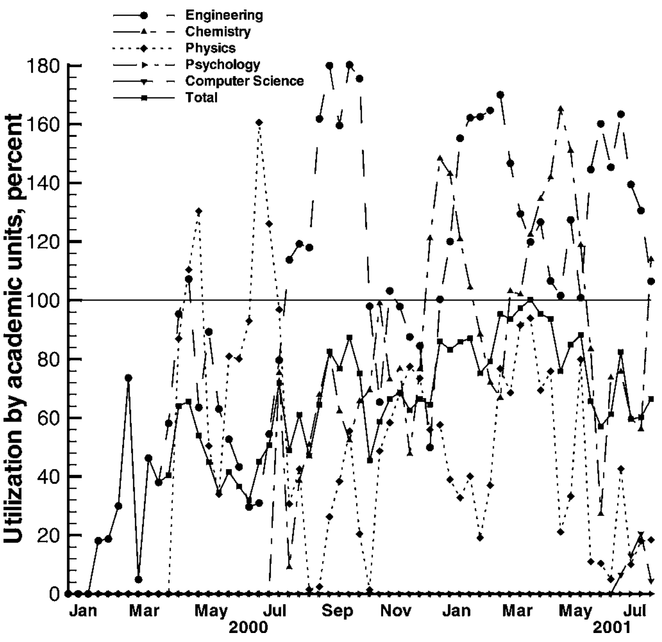

Figure 8. RCG utilization by users, 10 day increments.

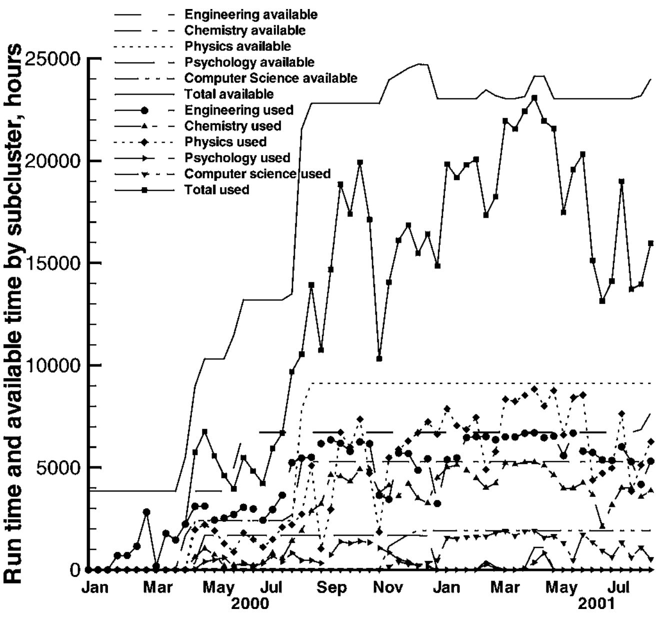

Figure 10. Utilization of individual subclusters and total available resources, 10 day increments.

ration), relatively few jobs were held in the queue waiting for the available processors. Table 2 illustrate an average queue waiting time for all and parallel jobs. It can be seen that more than 80% of all jobs are dispatched immediately (with waiting time less than two minutes) after entering the queue. Only 6% of all jobs are delayed for more than eight hours. The utilization level for individual academic units on the figure 8 appears higher than 100% at several periods, implying that the resources requested by this academic unit exceeded available local resources, and some of their jobs were transferred to the other subclusters of the grid. Figure 9 illustrates the cumulative utilization of the RCG through August 2001. It can be seen that run time of all jobs submitted by the more active academic units exceeds the total run time for all machines in their local subclusters, resulting in overflow jobs being sent to other academic units whose subclusters have CPU time available. This is one of the principal advantages of the RCG: it provides additional computing capability to academic units during periods of peak utilization by making available unused CPU cycles from the other academic units.

Unlike previously discussed figures, figure 10 shows runtime for individual subclusters regardless of which users utilize them. It also shows the theoretical upper limit of total available resources (see also table 1 for the hardware addition history of the RCG).[3] Note that due to episodic downtimes, routine maintenance, etc., the real resources availability would be somewhat less than the indicated theoretically available limit. Comparison figures 10 and 6 shows that

Table 3
Distribution of the job sizes.

Length of job (h)	Percent from all jobs submitted	Percent from total CPU time used
0–12	58.0	5.2
12–24	10.7	5.5
24–48	11.8	12.1
48–120	13.0	28.5
> 120	6.5	48.7

Table 4
Distribution of number of submitted jobs January 2000–August 2001.

Number of jobs submitted	Number of users
< 100	8
100–500	10
500–3000	8
> 3000	5

during high utilization periods the 'proprietary' principle becomes largely irrelevant. Due to the First Come-First Served nature of the job processing and random job execution duration, jobs have almost equal chance to be dispatched on any of the subclusters which would first have available nodes. Figure 10 also illustrate that Psychology and Computer Science subcluster were continuously utilized, despite the lack of local users submissions.

Figures 4 and 5 indicates the percentage of MPI parallel jobs submitted by each academic unit to the RCG and the time consumed by parallel jobs. It is seen that this percentage is gradually increasing, with MPI jobs taking approximately 50% of the total RCG runtime and constituting approximately twenty percent of all submitted jobs. Therefore, MPI operation represents a very significant part of the RCG utilization and justifies particular attention in planning and developing resource utilization facilities. User experience demonstrates that execution of MPI code on the RCG does not present a problem for the user. However, it was found that efficient processing of parallel codes requires creation of dedicated queues to allow intelligent use of the resources. Apart from designating machines on which MPI code could execute, such a queue should also automatically launch a system monitoring script to avoid the waste of CPU time in case of MPI code failure. It was found that the failure of one node executing an MPI code does not always lead to aborting of the code and other nodes continue to wait for the failing one until they are stopped by the user, which can result in significant waste of CPU time for large jobs. It was also found that MPI jobs are more sensitive to the quality of the RAM than single processor jobs. A significant number of failed MPI jobs were attributed to faulty memory chips.

Table 3 illustrate the distribution of jobs execution duration and a share of overall CPU time used by jobs of different duration.[4] It can be seen, that while the 58% of all jobs run with execution time less than 12 hours, they constitute only 5% of CPU time used. Almost half of total CPU time was used by the jobs which require more than five days (120 hours) to complete.[5] This conforms with the purpose of the computational grid to serve as a platform for the calculation which are impractical on the personal workstations.

Table 4 and figure 11 illustrate the distribution of number of jobs and percentage of a total RCG resources used by the active users as of August 2001. Note that both successfully completed and failed jobs are considered for average job run time calculation, where failed jobs encompass user code error, hardware problem during execution, MPI failure, deliberate termination of the job by user, etc. It is seen that the three most active users are responsible for more than 55% of all CPU resources used on the RCG, which prompt the introduction of fairshare in the new queue system. It can also be noted that the large percentage of runtime hours accumulated by the two out of four of the most active users have done so in significant extent due to extensive use of parallel applications. Table 5 and figure 12[6] shows average duration of the job for individual users. As can be seen, the majority of users utilize the RCG to run CPU intensive jobs rather than large amounts of relatively small jobs that could be executed at personal workstations, thereby confirming the paradigm of high throughput computing. Table 5 demonstrates that slightly less than a thirty percent of users utilize RCG for executing jobs with an average computation time below 12 hours, 12–24 hours and 24–48 hours, and 16% for jobs with average completion time above 48 hours.

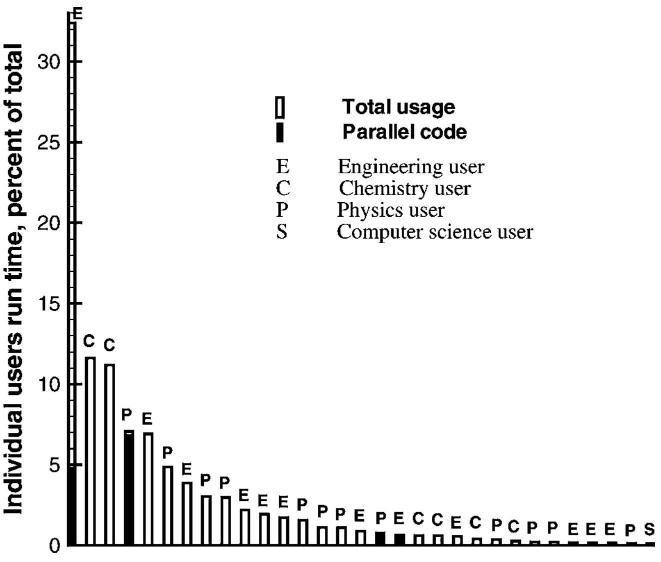

Figure 11. Percentage of the RCG resources used by individual users.

Table 5
Distribution of average job sizes (successful jobs January 2000–August 2001).

Average length of job (h)	Number of users
0–12	9
12–24	8
24–48	9
> 48	5

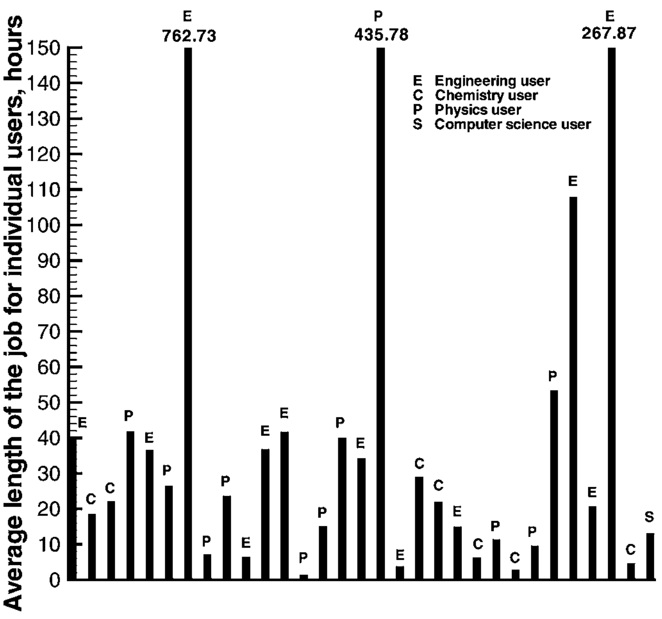

Figure 12. Average length of job for individual users.

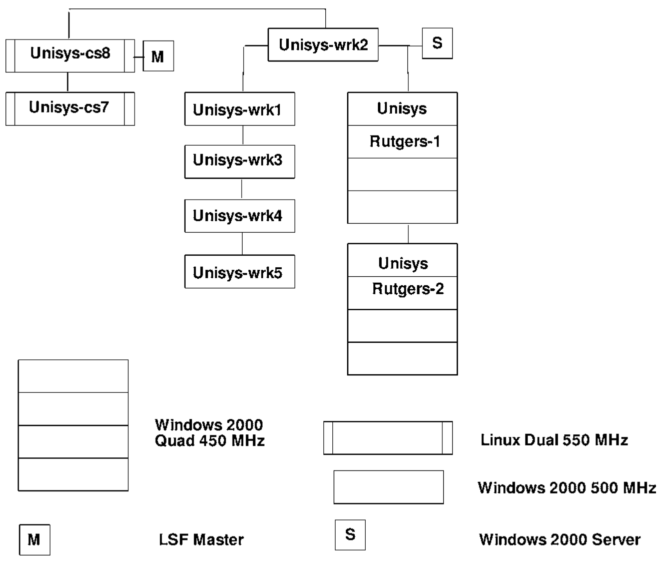

Figure 13. RCG-2 experimental cluster configuration.

5. Heterogeneous cluster

From the very beginning of the RCG project, one of the goals was to implement cycle scavaging, when jobs waiting in the batch queue would be able to use unused cycles on non-dedicated computers. Each large organization, such as university, possesses a number of computers, used only in an interactive mode and therefore used less than 25% of the time at best. It would be a highly attractive opportunity, if it would be possible to add such non-dedicated machines to the grid, to be used in the off-time hours. Despite attractiveness of such arrangement, there are several non-trivial issues which need to be resolved first. One of them concerns security consideration, since in the current form the RCG requires both NFS disk mounting and NIS user account sharing. Another problem is related to the necessity to implement checkpointing (which is useful, but not crucial feature for dedicated nodes) to allow jobs to be checkpointed and migrated to the dedicated cluster after primary owner would start interactive session.

Another important problem is that non-dedicated machines may have an operation systems which are different from the one used in the RCG. LSF is capable supporting cluster of heterogeneous nodes. To test the operation of such a heterogeneous cluster, the RCG-2, an experimental cluster consisting of two double processor nodes running the Linux operation system and 5 single and 2 quad processor nodes running Windows 2000 was implemented. The primary aim of this project was to gain experience of operation of the heterogeneous cluster and test its stability, as well as different approaches to its configuration.

The RCG-2 (see figure 13) was tested in February–May 2001. Different types of filesystem sharing, such as UNIX utilities for Windows, allowing standard NFS sharing and Samba were tested. The cluster demonstrates practical capability of the LSF to handle a heterogeneous cluster and provides reasonably stable operations. The planned integration of RCG-2 into the RCG at the current moment is delayed due to the necessity to upgrade the RCG LSF from version 3.2 to 4.1, supporting Windows 2000 operation system, which would require temporary shutdown of the grid.

6. Conclusion and future work

The RCG demonstrates the validity of the computational grid concept in academic environment. The RCG was created to provide faculty and students of participating academic units of Rutgers University with a shared computational resources for high throughput computing that could be utilized in a flexible and efficient manner. The RCG began operation using LSF in January 2000. The RCG currently includes 58 dedicated dual processor computers combined in five subclusters. It has proven to be a highly reliable system, with low downtime and high utilization and in a high demand among students and faculty of participating academic units. The utilization level of the grid approaches 80%, while utilization by some academic units exceeds the maximum available on their local subcluster. The excess jobs are automatically dispatched to the available hosts at other academic units, successfully realizing the principle of dynamic distribution of the jobs according to the load on a local subcluster. During the first 20 months of operation, 105 users accumulated 731,188 calculation hours and 1,000,000 hours mark was exceeded in March 2002. Parallel jobs employing MPI are successfully executed on the RCG and constitute a noticeable share of the total RCG load.

The RCG has enabled new scientific research to be accomplished. Users were utilizing the RCG to obtain a scientifically important results since the beginning of the pilot phase of the project, with twelve conference and journal publications [19] published with the results obtained on the RCG during the pilot phase of the project (January–October 2000) alone. As a result, the RCG was considered mature enough to change status from pilot to production mode. Since that time, a large amount of users applied and obtained accounts on the RCG, resulting in high utilization levels. Due to this increase in the number of users in the production phase, a priority and a fairshare system was implemented in AY2001-2002 to regulate the distribution of the resources.

The future development of the RCG will be conducted along two primary directions. The first is aimed at continuing improving queue system flexibility and user support. This would include creating and maintaining a more diverse and efficient series of queues aimed at supporting specific requirements of individual users or group of users and maintaining the maximum efficiency of CPU resources utilization and allowing selection of hosts best suited to a particular user's problem as discussed above. An example of such system of wrappers and scripts is described in [14]. A more comprehensive system of scripts and wrappers would be implemented to facilitate batch operations and made them more transparent for users.

The second direction of effort will be focused on expanding the capability of the RCG by implementing resource scavaging in addition to the cluster of dedicated computers. By adding personal machines to the common pool during off-time hours, the available computational resources can be increased several fold, without acquisition of new computer systems. With the heterogeneous operation successfully tested with the aid of the RCG-2 experimental cluster, the possibility is open to add non-dedicated machines with diverse operation systems. The next step, necessary for cycle scavaging implementation would be the implementation of job migration which would allow checkpointing and preempting. This is necessary since the primary owner of a non-dedicated computer(s) must be able to migrate the current job out of owner' machine without causing its termination. This requires, however, code recompilation against LSF specific libraries, which might not be convenient for all users. Users will be given the choice between waiting in a queue in the dedicated cluster or being able to access non-dedicated machines at the expense of having to recompile their codes to include LSF libraries. The addition of machines with different operation systems would also require development of a more comprehensive set of wrapper codes, providing user with transparent and system-independent environment.

Acknowledgment

The authors wish to acknowledge the support from Rutgers Strategic Resource Opportunity Assessment (SROA) program.

Notes

1. While such modification could seem trivial, experience shows that the instances of code failure caused by incorrect submission procedure remains rather high if any additional options need to be used in the submit command. It is therefore recommended to implement as many necessary options in the queue as possible.
2. We consider user as active, if he or she used the RCG for at least one thousand CPU hours for production calculations between January 2000 and August 2001. Users requesting but not activating accounts or running only test job for system familiarization with total CPU time below one thousand hours are not considered in this paper.
3. Psychology subcluster was inactive since 25 December 2000, with the short uptime in April 2001. It was upgraded to 8 dual processor nodes and return to operations in October 2001.
4. To avoid distortion of the distribution by the jobs failed immediately after dispatch, jobs with execution time below 5 minutes were not considered.
5. For parallel jobs the execution time is a sum of CPU time used by all requested processors.
6. The individual users on the figure 12 correspond to the individual users on figure 11.

References

[1] R.J. Allan, Survey of Computation Grids, Meta-Computers and Network Information Tools ed.2, CLRS Daresbury Laboratory, Daresbury, January 2000.

[2] S. Anasiadis and K. Sevcik, Parallel application scheduler on network of workstations, Journal of Parallel and Distributing Computing 43 (1997) 109–124.

[3] T.E. Anderson, D.E. Culler, D.A. Patterson and the NOW Team, A case for networks of workstations: NOW, IEEE Micro (February 1995).

[4] R.H. Arpachi, A.C. Dusseau, A.M. Vahdat, L.T. Liu, T.E. Anderson and D.A. Patterson, The interaction of parallel and sequential workloads on a network of workstations, in: *Proceedings of ACM SIGMETRICS'95/PERFORMANCE'95 Joint International Conference on Measurement and Modeling of Computer Systems*, May 1995, pp. 267–278.

[5] A.C. Arpaci-Dusseau and D.E. Culler, Extending proportional-share scheduling to a network of workstations, in: *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)* (1997).

[6] P. Boulet, J. Dongarra, F. Rastello, Y. Pobert and F. Vivien, Algorithmic issues on heterogeneous computer platforms, Parallel Processing Letters 9(2) (1999) 197–213.

[7] R. Buyya, *High Performance Computer Clusters: Architecture and Systems*, Vol. 1 (Prentice-Hall, NJ, 1999).

[8] J. Dongarra, An overview of computational grids and survey of a few research projects, in: *Symposium on Global Information Processing Technology*, Japan, 1999.

[9] X. Du and X. Zhang, Coordinating parallel processing on network of workstations, Journal of Parallel and Distributing Computing 46 (1997) 125–135.

[10] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure* (Morgan Kauffman, San Francisco, CA, 1999).

[11] V. Hamscher, U. Schiegelshohn, A. Streit and R. Yahyapour, Evaluation of job-scheduling strategies for grid computing, in: *1st IEEE/ACM International Workshop on Grid Computing*, Bangalore, India, 17–20 December 2000.

[12] G.J. Henry, The fair share scheduler, AT& T Bell Laboratories Technical Journal 63(8) (October 1984) 1845–1857.

[13] http://www.dhpc.adelaide.edu.au/education/honours/campusgrid.html

[14] http://www-isd.fnal.gov/fbatch

[15] J. Kay and P. Lauder, A fair share scheduler, Communications of ACM 31(1) (1988) 44–55.

[16] D. Knight, G. Zhou, N. Okong'o and V. Shukla, Compressible large eddy simulation using unstructured grids, AIAA Paper No. 98-0535, 1998.

[17] Load Sharing Facility, http://www.platform.com

[18] MPICH: A Portable Implementation of MPI, Mathematics and Computer Science Division, Argonne National Lab, http://www-unix.mcs.anl.gov/mpi/mpich/index.html

[19] Rutgers Computational Grid, http://coewww.rutgers.edu/rcg/

Boris Chernyavsky received his Ph.D. in aerospace engineering from Rutgers – The State University of New Jersey in 2002. His research interests include application of Computational Fluid Dynamics (CFD) to solving problems of compressible gas dynamics, turbulence modeling, reacting flow modeling and combustion. His computational interests include cluster and grid computing, load balancing and scalability issues for highly parallel computing and scheduling and dynamic resource allocation algorithms. He is currently holding a position of Post-Doctorate Fellow in Georgia Institute of Technology.
E-mail: chernyavsky.boris@ae.gatech.edu

Emilio Gallicchio holds a Ph.D. degree in chemical physics from Columbia University (1996). Since July 2000 he is the Associate Director of High Performance Computing at Rutgers University. His research interests include the solvation thermodynamics of biological macromolecules, protein structure prediction and ligand binding affinity prediction. His computational interests include the design of fast continuum solvent algorithms for macromolecules, parallel and distributed free energy perturbation schemes and the deployment of computational clusters for large-scale parallel computations. He is the author of a book, review papers and numerous research papers. He is one of the original authors of IMPACT, a large and complex software package for molecular simulation of biological macromolecules.
E-mail: emilio@hpcp.rutgers.edu

Doyle Knight received his Ph.D. in aeronautics from the California Institute of Technology in 1974. Following two years service in the United States Air Force as an Aeronautical Engineer, and one year Postdoctoral Fellowship in applied mathematics at the California Institute of Technology, he joined the faculty of the Department of Mechanical and Aerospace Engineering at Rutgers – The State University of New Jersey. His research interests focus on the application of Computational Fluid Dynamics (CFD) in compressible flow and design optimization. His research in compressible flow has extended to a broad range of topics including shock wave turbulent boundary layer interaction, incipient separation on pitching airfoils, turbulence model development, high speed inlet unstart and effects of unsteady energy deposition in supersonic flows. His research activity in design optimization focuses on the application of CFD to the automated optimal design of high speed air vehicles (especially, missiles). His research interests have led to his significant activity in parallel processing on PC clusters under Linux including the development of Linux clusters in his research laboratory.
E-mail: ddknight@rci.rutgers.edu

Ronald M. Levy received his Ph.D. in biophysics from Harvard University in 1976. In 1980 he joined the faculty at Rutgers University where he is currently Professor of chemistry and chemical biology and Co-Director of the Institute for Biology at the Interface of the Mathematical and Physical Sciences (BioMapS). His research interests are in the following areas: biophysical and physical chemistry of proteins and aqueous solutions, liquid state chemical physics, development and application of computer simulation methods for studying large molecules, protein structure, dynamics and folding, statistical thermodynamics of ligand binding, high performance computing in biophysical research, functional genomics and bioinformatics.
E-mail: ronlevy@lutece.rutgers.edu

Andrew Page received his B.S. in physics from Rutgers University in 1995. Following three years as a faculty member of New Brunswick High School, he returned to Rutgers to pursue his interests in instructional computing. His work focused on automated system deployment, instructional technology, and LEGO robotics. He currently serves as Manager of Network Services, for the Telecommunications Division at the university. His group has developed applications for the management of residential networking, network management and modeling, traffic shaping, and distributed permission models. He also serves as an instructor for Rutgers University Center for Applied Computer Technology, where he teaches classes in Oracle database programming and administration.
E-mail: apage@td.rutgers.edu