Profiling Scheduler for Efficient Resource Utilization

Alexander Bogdanov[1], Vladimir Gaiduchok[2(✉)], Nabil Ahmed[2], Amissi Cubahiro[2], and Ivan Gankevich[1]

[1] Saint Petersburg State University, Saint Petersburg, Russia
[2] Saint Petersburg Electrotechnical University "LETI", Saint Petersburg, Russia
gvladimiru@gmail.com

Abstract. Optimal resource utilization is one of the most important and most challenging tasks for computational centers. A typical contemporary center includes several clusters. These clusters are used by many clients. So, administrators should set resource sharing policies that will meet different requirements of different groups of users. Users want to compute their tasks fast while organizations want their resources to be utilized efficiently. Traditional schedulers do not allow administrator to efficiently solve these problems in that way. Dynamic resource reallocation can improve the efficiency of system utilization while profiling running applications can generate important statistical data that can be used in order to optimize future application usage. These are basic advantages of a new scheduler that are discussed in this paper.

Keywords: Computational cluster · Scheduler · HPC · Profiling · Resource sharing · Load balancing · Networking

1 Introduction

Clusters for scientific calculations became widespread during the last decades. Initially, they were specific to government laboratories, large companies or major universities. But nowadays many non-commercial organizations even with a limited budget can afford a computational cluster. Such clusters usually consist of many homogeneous, relatively inexpensive nodes. This approach became quite common for small universities as well as for prominent IT companies: it allows organizations to reduce total cost of ownership and facilitate administrative tasks. Such systems are much cheaper than big SMP nodes. The same can be said about hardware upgrades due to the fact that nodes consist of common, widespread components and many manufacturers produce it. Actually, this fact means that organizations will probably not get vendor lock-in. The next advantage is scalability. SMP systems are known to have limited scalability while clusters can be easily scaled up to thousands of cores.

The same situation can be found in the supercomputer area. The evolution of the TOP-500 list is an illustrative example. If in 2000 (June 2000 list) only 2.2% of the systems have the cluster architecture (providing 3.5% of overall

O. Gervasi et al. (Eds.): ICCSA 2015, Part IV, LNCS 9158, pp. 299–310, 2015.
DOI: 10.1007/978-3-319-21410-8_23

performance of the systems in the list), in 2014 (November 2014 list) 85.8% of systems were clusters (providing 67% of overall performance) [1].

This paper concerns effective computational cluster utilization. It implies that organization will not get resources underloaded or overloaded while users will be able to compute their tasks fast.

2 Computational Infrastructure Problems

Despite the fact that clusters provide users and administrators with many advantages which other architectures have never had, one can face with many issues while creating, maintaining and using a computational cluster. Sometimes it is really difficult to create a reliable computational infrastructure. Some major concerns and problems of the cluster architecture are listed below.

– **Planning.** An HPC organization should pay much attention to infrastructure planning and upgrading. Decisions at this stage determine how flexible and reliable the computational infrastructure will be. There could be many questions. How many cores per node will be acceptable (should we use big SMP nodes)? How much RAM should be installed (a compromise between requirements and cost)? Which vendor offers best price to performance ratio? Should accelerators (e.g. GPGPU) be used (will user programs harness all GPGPU powers)? There is a tendency in the HPC world to use hybrid clusters for scientific computations [2]. Nowadays (the data from the last TOP-500 list, November 2014), 15% of the TOP-500 systems are equipped with some accelerator [1]. And 34.5% of the TOP-500 overall performance is contributed by hybrid systems (one can compare the present situation with the situation in 2009 when systems with accelerators contribute only about 5% of the TOP-500 overall performance). Moreover, some clusters have several accelerators per node (e.g. multi-GPU nodes) as well as nodes with different accelerators (although they usually belong to different clusters). There are several kinds of computational accelerators. It can be a GPGPU, MIC or some other accelerator. Such hardware is used in order to increase the system performance: accelerators can speedup scientific applications; they usually contribute a high percentage of the peak performance of the hybrid system. But sometimes they just can not help. The first possible problem is the algorithm being used: if it can not be parallelized or a program that implements it has a substantial sequential fraction, the manycore device will be underloaded almost all the time of a program run. One should always remember Amdahl's law. But even in case of good algorithm (in terms of parallelization) there can be other issues. For example, data transfer from the RAM to the device memory and vice versa. It can be a possible bottleneck. Despite the fact that contemporary accelerators can give a substantial speedup, one should carefully estimate the possible benefits. Are there enough software that supports GPGPU? Will users run it? Which algorithms does it use? Answering these questions will help to make the decision on the hybrid systems usage.

- **Data Storage.** Should it be a local storage, SAN or NAS? Which protocols to use? Should a parallel file system be used? One should think about many factors while planning the data storage (e.g. data consolidation). The final goal is to alleviate all possible bottleneck which can be caused by slow reading/writing and provide users with comfortable and transparent access to their data. The specific solutions depend on computational center objectives, specialization and user tasks. Different computational centers work with different amounts of data. Data sets differ too. The most challenging case is working with big data, but not every center will face it.
- **Networking.** Cluster architecture offers great scalability, but introduces some new problems, one of which is networking. Networking can become a bottleneck. One of the main factors is the ratio of CPU performance to networking performance. Increasing the number of processes that participate in a computations will not lead to speedup if the cluster network has low performance. Moreover, it could lead to slowdown. In order to alleviate this problem one should use algorithms with as few synchronization as possible. But such requirement could not be suitable taking into account great variety of user applications. Custom interconnect is a big part of the cost of a supercomputer. But even small computational center can consider networking as a very important factor in the effectiveness of its work. It is advisable to use high performance networking for clusters. There can be two separate networks: a network for management and maintenance tasks (e.g. Ethernet 1G) and separate computational network (e.g. Infiniband FDR). Some aspects of networking in computational center are discussed later in this paper.
- **User Friendly Interface.** Not every user is a computer science specialist, they just want to calculate their tasks (e.g. scientific calculations in case of university computational center). Nodes of computational clusters are usually installed with some Linux distribution. Traditional cluster management systems (e.g. some implementation of PBS) are usually have text interface as a default or the only interface, but inexperienced users are not accustomed to work with the console. So, it takes time for them to learn and get accustomed. It is not convenient neither for users nor for organization because users can do a lot of mistakes at that time (while working with the clusters). In order to solve this problem organization can install some additional packages that will ease task submission (e.g. web interface). It can also give users virtual machines. It is more convenient and secure than single server that provide users with an access to computational resource. While virtual machines can be used as computational resources too. Such approach can be very convenient and beneficial for users as well as for organization [3]. There is also another big problem: regardless of the interface, users should choose appropriate resources for their tasks. Making a decision on computational resources can be difficult. Requesting too much resources could lead to poor cluster utilization. But users want to calculate their tasks fast. So, organization should find a compromise solution that is acceptable for different users, provides fair resource sharing and high resource utilization. That is why the

following question becomes one of the crucial questions for computational center.
- **Resource Sharing, Resource Monitoring, Resource Utilization.** These are the main questions of this paper, they are discussed below.
- **Security Questions.** Security issues is very important too, but they are out of scope of this article.
- **Other Questions.**

3 Resource Sharing

HPC resource in general can be shared by different groups of users that use open and closed source software packages, that in turn use various standards which makes efficient resource sharing a challenging problem. One should take into account every scientific application separately. Solving this problem seems to be impossible without intelligent software.

One of the possible solutions is to use a profiling scheduler. Such scheduler can collect information about running programs. These information reflects usage efficiency and can be used for future application run.

All in all, we can distinguish the following approaches that are used in order to share computational resources.

- **No Management System.** One can install operating system on computational nodes and grant access to nodes to users. It is the simplest approach from infrastructure point of view, but it is difficult to do resource planning in the described system.
- **Single System Image.** There are several packages that allow administrator to create such system. Several computational nodes will be viewed as a single system. Such systems are quite convenient but they also have drawbacks: computational resources at the physical level are separate, so questions on scaling up emerge. Users do not pay attention to this fact, so they can use some API for SMP systems and get small speedup or even slowdown. Another possible problem is failure of one node, because it could lead to the restart of the whole system.
- **Cloud.** There are several well-known and reliable solutions that can be used in order to create cloud. Many organization uses this approach due to the advantages that it offers (e.g. relatively simple infrastructure management). But such approach implies virtualization overheads. In addition, there are still some questions about load balancing (e.g. virtual machines migration).
- **Classical Management System.** We consider portable batch system (TORQUE, PBS Professional) and similar systems (HTCondor, Univa Grid Engine) as classical because they offer a traditional approach and are widely used in many computational centers [4] [5]. This approach is a base for a new one that will be discussed in this paper. Such system will be described in the next chapter.
- **Other Approaches.**

4 Traditional Approach

Such approach implies reservation of available computational resources: users request some resources by submitting their jobs. We will discuss this approach on the example of PBS system.

The key notion of PBS is a queue. Cluster nodes are assigned to some queue. There are usually several queues in such systems. One node can belong to several queues at the same time. Queues have different parameters: priority, resource limitations, time limitations, and so forth. Cluster administrator declares some policy for job queuing. According to this policy, he creates queues and assign necessary parameters to them. For example, he can create a queue for fast jobs with weak resource limitations and queue for jobs that requires a long time to complete with severe resource limitations or low priority. The common PBS system can be viewed as shown in Figure 1.

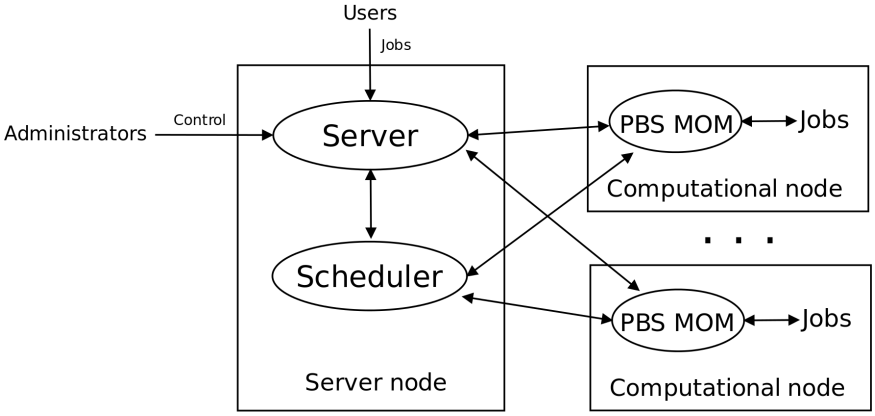

Fig. 1. PBS Scheme

Many organizations across the world use some PBS implementation or similar system to organize resource sharing. PBS provides managers and administrators with relatively simple solution in terms of resource sharing policies as well as administration efforts. It allow users to submit their jobs and specify resource requirements for them.

The PBS scheduler decides at which time and on which resources a job will be run. When it is possible (according to the current resource reservation and policies) the scheduler informs the PBS server. The PBS server, in turn, starts the job on allocated resources. Since that moment the resources that were allocated for the job are reserved. No one else can use those resources (even another job of the same user).

Such approach is quite clear and simple, but it has a drawback. The PBS usually does not monitor dynamic resource load. It monitor available resources only in terms of reservation and requests. The PBS system relies on resource

characteristics that were initially assigned by the administrator. Such resources as CPUs, GPUs, the overall amount of RAM will rarely be changed. And when there was some hardware upgrade the system administrator can specify new resource characteristics for PBS. But one can not retrieve the information about the actual resource load. The only thing that a PBS implementation depicts is jobs and resources assigned to it. One can not figure out the current load of nodes (e.g. load average value) using PBS utilities. One can assume that the amount of resources reserved for a job reflects the actual resource usage. But it could not be the case, for example, when a user application that is started within the job is not parallelized very well, or user made a mistake when writing that job. So, when administrator use the PBS he relies on users: users should submit correct jobs and predict the best resource amount in order to reserve only really necessary resources.

The other drawback is the fact that an average PBS implementation provides administrators and managers with scarce accounting information. One can find out only overall resource usage (e.g. CPU time) and only when job is finished. One can not find the detailed information, for example, load on different nodes that participated in computations. Moreover, that scarce information is usually stored simply in log file along with other log messages. If administrator wants to get more user friendly interface he should write it (e.g. some script) or use third-party one. Finally, there is no reliable way to determine particular applications that were executed inside job script. This is the another case when administrator relies on users: they should name their jobs appropriately in order to reflect which application is used within a job. But, of course, users can name their jobs in some different way, the way they think to be the most convenient for them. So, the name of a job is not a reliable source of information. Moreover, users can run several different applications within a single job.

The notion of a job is user-defined anyway. A user determines what will compose a single job. But the organization managers want to know application usage information. For example, they want to know which application is the most frequently used or which one has the worst CPU time to wall clock time ratio. Such information can be very useful for planning the HPC infrastructure (for example, when planning some hardware upgrades or purchase of some new software licenses). The first problem is solved when software package comes with a license server. The administrator can estimate the application usage from the license server log file (calculate the number of license checkouts). But proprietary software may have no license server (for example, some paid closed source scientific packages have no license server), let alone open source software. So, administrator just can not count the application usage in this case. They can write some sophisticated prologue script (in terms of PBS) which will try to find out the job content, but such approach is difficult and not quite reliable, because a job script can call many other scripts and depends on environment variables (for example, a string with application call without absolute path in a script can actually run different versions of the application depending on PATH variable). It is really hard to find out which versions of which applications will be executed

in the job in this case. So, standard PBS utilities of an average PBS implementation can not determine which applications are executed within the job while such information can be very helpful for the future HPC infrastructure planning and for the annual reports. Of course, profiling can be done using third-party software (not by a PBS implementation), but such software will not be able to map resource usage to PBS jobs.

The detailed accounting information could be used in another way. One can analyze it in order to improve the overall resource usage. Two similar jobs can show different performance despite the fact that user requested the same resources for them. Despite the fact the CPU cores count was the same the first job, for example, could be executed on a single node while the second on separate nodes in the network. Or different nodes were assigned in the first run and in the second run. And even in case of exactly the same allocated resources it is possible to get different performance because of shared resources like network.

Accounting information can also give the representation about the underlying levels performance (underlying levels that some application uses). As an example, MPI implementation. There are many different MPI implementations that varies greatly in performance: one of them can support Infiniband RDMA and another (probably old version) can not (work only with IP over Infiniband). As a result, the same application run can yield different performance when using different dynamic MPI libraries. Such situation can be figured out by profiling applications.

Figures 2 and 3 illustrate test runs of an MPI application (Intel IMB test PingPong) that were executed on two separate clusters (see Table 1) using TCP/IP (over Ethernet), IPoIB (IP over Infiniband) and RDMA protocol (Infiniband).

Table 1. Cluster characteristics

	Cluster 1	Cluster 2
CPU	2x Intel E5335 2.0 GHz	2x Intel X5650 2.67 GHz
GPGPU	-	3x (8x) NVIDIA Tesla M2050
RAM (GB)	16	96
HDD (GB)	160	120
Network	Ethernet 1G, Infiniband 4x DDR	2x Ethernet 1G (bond interface), Infiniband 4x QDR
Total characteristics	768 Tb RAM, 48 nodes, 384 cores	2.3 Tb RAM, 24 nodes, 288 cores, 112 GPUs
Peak performance (Tflops)	3.07	59.6

As one can see, using RDMA is much more beneficial than IPoIB. Infiniband RDMA is a special protocol that was designed for Infiniband systems (details on working with Infiniband could be found in many sources, e.g. [6]). But RDMA should be supported by MPI implementation. So, planning the computational

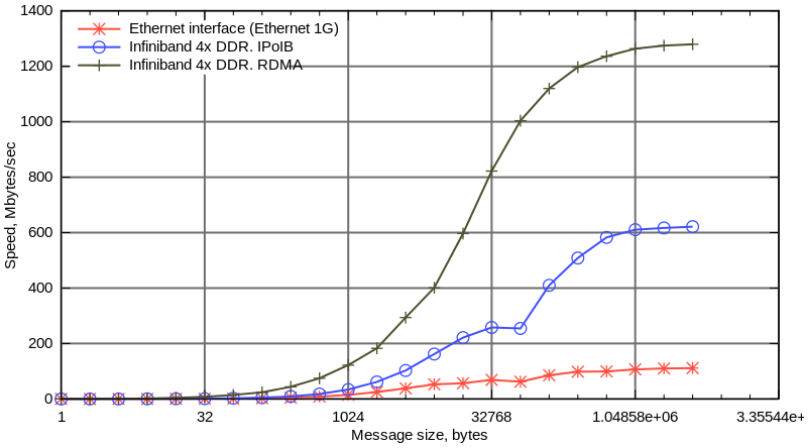

Fig. 2. Network test on the cluster 1

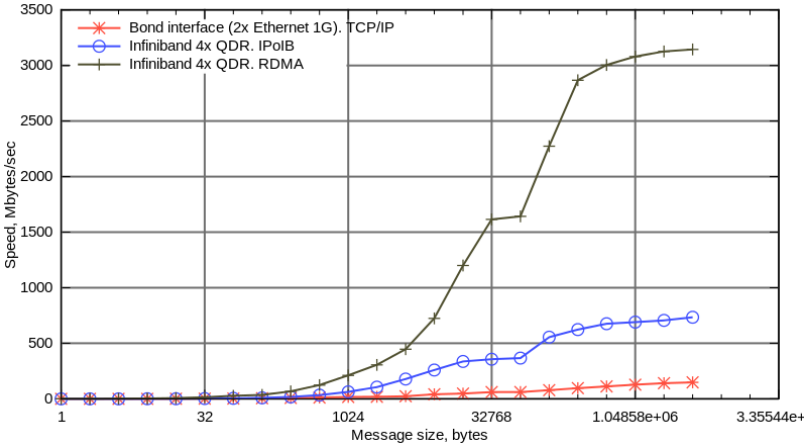

Fig. 3. Network test on the cluster 2

infrastructure is not limited to hardware, it also concerns system software, as well as application software.

All in all, PBS promotes a way of conventional resource sharing that is quite simple and clear. But such approach works efficiently only under certain assumptions. It does not handle possible user errors efficiently: one can imagine the situation when user requests 128 cores for his job, but the job uses only one CPU core due to some mistake, let alone wrong estimations when users can not predict correctly the necessary amount of resources (for example, inexperienced user requests many cores for a serial job). The opposite situation is one when

user application utilize more resources than it was requested. Such situations could be find out using detailed monitoring or profiling which is not specific to the PBS. As a result computational resources can be underloaded or overloaded. The new scheduler was designed in order to solve these problems.

5 New Approach

The new approach is based on profiling. It provides administrator with detailed accounting information about running jobs. The information retrieved from the profiling is used in order to dynamically reallocate the available resources. The main purpose of creating this scheduler is to overcome PBS drawbacks while trying to keep the simplicity of administration and utilization and do not introduce substantial overheads.

This scheduler is designed to be modular. The work is still in progress, we implemented basic functionality and tested it on several nodes at our faculty. Here are the basic features of the scheduler.

- **Profiling, Detailed Accounting and Monitoring.** All jobs are traced by the corresponding module (accounting module). This module starts the job and captures events of new process creation (using fork, vfork or clone system call), stop or execution (when a new binary starts within the process). The main idea is to gather statistical data about each process of the job (binary that was executed, CPU time, memory usage etc.). One can consider it as unnecessary, but such approach allows one to analyze user jobs and find possible bottlenecks. Moreover, statistical data can be used for predictions. Only a few programs perform the analysis of their run. We can collect performance data without the necessity to go deeply into the source code or trace each program step. So, such accounting is suitable for open source programs as well as closed source ones.
- **Dynamic Resource Allocation.** Our system provides flexible reservation of all available resources. This means that the amount of resources user requests will be treated as hints, not precise numbers. Our system will monitor the jobs and check whether jobs meet the requirements specified by users. A proper computational job should correspond to the resources requested for it. But an average job can underload the requested resources or overload them. Using conventional PBS one will have free unused resources in case of underload, and users which jobs are still in a queue. While in case of overload one will have conflict of interests resulting in increase of job completion time. In the both cases one have inefficient resource utilization. Our system will start another job on the underloaded resources (which requirements meet the hardware characteristics) while giving the initial job higher priority (if resources become overloaded by these two jobs, the second one will be suspended). In case of overload our system offers to administrator three possible options: stop the job and send an appropriate message to the user, suspend this job and run another or keep the job running and log this information.

Such flexible approach allows administrators to alleviate user mistakes and improve resource utilization.

- **Rating.** Our system will have a module for keeping user rating. We propose this simple feature which is rarely used when we talk about clusters for scientific computations. User rating will reflect the efficiency of user jobs. Experienced user can estimate the necessary resources for his job, so running such job will not lead to underload or overload. If some job consumes more resources than it was requested, the rating of the job owner will be decreased. And if some user requests more resources than he actually needs, the rating of such user will be decreased too. The goal of such rating is to improve the priority and limitation policies: rating will impact jobs priority and resource limitations for each user. Detailed accounting, once again, can help users to find the possible problems of their jobs.
- **Predictions.** The predictions module can automate resource requests. Such module will offer to user possible amount of the resources that is necessary for his job or even make such requests instead of user (user will have to specify the job only). Of course, the performance of an application depends on the exact task (input data). But one can find out the general tendencies that are specific to the application. For example the fraction of the sequential part will impact the highest speedup that can be achieved (Amdahl's law). The underlying levels can impact the performance too. One can remember the example from the previous section that depicts MPI implementation features. Prediction module can estimate the necessary resources in accordance to the used libraries. And finally it can mark the applications that can use specific hardware (GPGPU) in order to run such applications on the necessary nodes. These can be figured out using statistical data that is gathered by the accounting module. Of course, there should enough statistical data in order to make the correct prediction.
- **Native API.** The accounting module uses ptrace in order to trace and suspend jobs. When this module starts it forks and the newly created process calls ptrace (specifying PTRACE_TRACEME), then it runs the necessary application (by execve system call). The tracer is notified when tracee creates a new process (using fork, vfork or clone), execute another program (using one of exec* functions; it is required in order to get the information about actually running programs) and when one of the tracee processes exits (for retrieving accouting information). The accounting information about process is gathered from the procfs. Another important API that is used in our system is Pthreads. Invocation of the accounting module on a remote node is done by changing the ssh command. Such calls will be traced and the command for the remote node will be changed in order to invoke the accounting module (it is necessary to monitor the jobs resource usage). So, such approach could be considered as a native to Linux systems.
- **Testing.** The dynamic resource allocation is also supported by periodic tests. They are executed in order to reflect the actual resource state (for example, the network load). Using such information the scheduler can allocate the best nodes (in terms of the performance) for new jobs.

– **The Possibility to Use the Modules in Different Ways.** For example, accounting module can be used in a PBS cluster. In this case, administrator should add an invocation of the accounting program before actual user script invocation in the PBS prologue script. Administrator should pass user script with all the options as an arguments to the accounting module.

Testing of our system shows that overheads imposed by profiling are smaller than 1% when compared with PBS implementation on a proper job test set. While an average improper job test set (that underloads and overloads the resources) was computed about 16% faster. The proposed system can be viewed as it depicted in Figure 4.

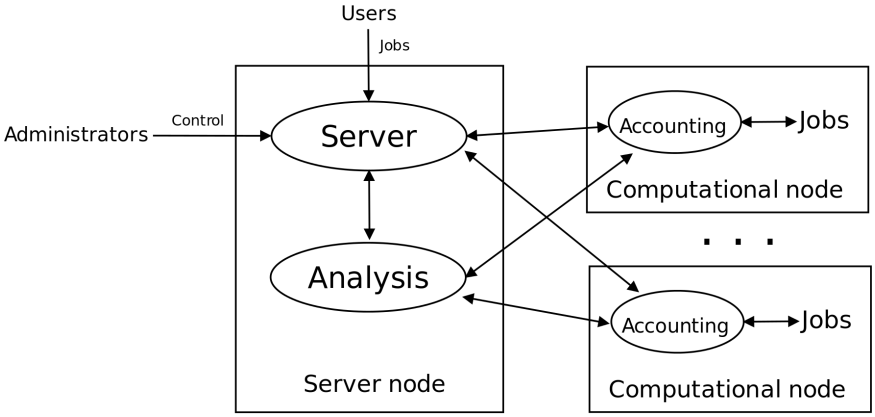

Fig. 4. Scheduler scheme

Such system performs the constant detailed monitoring which generates the accounting information that helps to make real time decisions on resource reallocations (in case of underload or overload) while is also saved for the future use. The rating module will use this information for keeping the user rating that will encourage users to request resources correctly. The prediction module will use this information in order to offer the optimal amount of resources to users and probably automate this task in the future.

Such approach can ease the administration tasks as well as utilization of computational clusters. It could optimize the resource usage and helps to make efficient decisions on computational infrastructure questions.

6 Conclusion

Planning infrastructure for scientific computations sometimes can be challenging. One should take into account many factors. User jobs can underload or overload requested resources, that is why administrator should use systems that

will monitor the situation. Statistics generated from such monitoring can be used for future application run.

Scheduling applications can be done using the approach described. Each job submitted to the system is profiled and its optimal resource reservation is determined. For subsequent submissions this reservation is used by default.

Basic modules are implemented using native to Linux APIs and instruments. There can be several modules that perform predictions based on statistics, test actual resource parameters, calculate user rating. When finished, these modules will ease administrative tasks as well as user work.

Acknowledgments. The research was carried out using computational resources of Resource Center Computational Center of Saint-Petersburg State University within frameworks of grants of Russian Foundation for Basic Research (project No. 13-07-00747) and Saint Petersburg State University (projects No. 9.38.674.2013 and 0.37.155.2014).

References

1. The TOP-500 list. http://www.top500.org/statistics/list
2. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Kru ger, J., Lefohn, A.E., Purcell, T.: A survey of general-purpose computation on graphics hardware
3. Gayduchok, V.Yu., Bogdanov, A.V., Degtyarev, A.B., Gankevich, I.G., Gayduchok, V.Yu., Zolotarev, V.I.: Virtual workspace as a basis of supercomputer center. In: 5th International Conference on Distributed Computing and Grid Technologies in Science and Education, pp. 60–66. Joint Institute for Nuclear Research, Dubna (2012)
4. TORQUE, Adaptive computing. http://www.adaptivecomputing.com/products/open-source/torque
5. PBS Professional, Altair PBS Works. http://www.pbsworks.com/Product.aspx?id=1
6. Mellanox OFED User Manual. http://www.mellanox.com/page/products_dyn?product_family=26