Xen and the Art of Cluster Scheduling

Niels Fallenbeck, Hans-Joachim Picht, Matthew Smith, Bernd Freisleben
Department of Mathematics and Computer Science, University of Marburg
Hans-Meerwein-Strasse, D-35032 Marburg, Germany
Email: {fallenbe, picht, matthew, freisleb}@informatik.uni-marburg.de

***Abstract*— In shared use clusters, scheduling systems must schedule both serial and parallel jobs in a fair manner, while at the same time optimizing overall cluster efficiency. Since serial and parallel jobs conflict considerably, scheduling both types of jobs concurrently is a difficult task. Two major strategies are in common use: partitioning the cluster (thus avoiding the problem) and reservation combined with backfilling. Both have major drawbacks in overall performance, ease of use and fairness depending on the particular configuration, which can lead to heated debates between the users of the different types of jobs. In this paper, we introduce an approach to create dynamic virtual cluster partitions using para-virtualization techniques, to deal with the conflicts between parallel and serial jobs. The system dynamically adjusts to different types of job loads and offers easy and transparent use and configuration to both users and administrators, without resorting to complicated runtime prediction or backfilling algorithms. A proof-of-concept implementation based on the Sun Grid Engine scheduling system and Xen is presented.**

***Index Terms*— Cluster Computing, Virtualization, Scheduling, Parallel/Serial Jobs, Backfilling**

I. Introduction

High-performance computing is becoming increasingly important in many industrial and scientific application areas. However, high-performance computing clusters are relatively expensive for small/medium enterprises and individual research groups in universities, resulting in cross-organization purchases and shared use scenarios.

This leads to different types of applications (serial and parallel) to be executed within a single cluster. Typically, a cluster scheduler is responsible for ensuring fair use and maximum system workload and must deal with different types of jobs. However, most scheduling systems are designed to prefer particular classes of jobs (i.e. parallel, serial, long running, short running,...). As a result, cluster owners or administrators are sometimes forced to partition their resources into separate sets of computing nodes, e.g. one sets that is dedicated to parallel applications and one set that is used for serial applications. If there is an uneven distribution of the different job classes on the whole or over time, this leads to unnecessary idle time, since jobs from one partition cannot easily be moved to the other partition [1]. With the advent of Grid computing, this problem is amplified, since it becomes more difficult to predict when what type of job will be submitted to a cluster, which in turn makes it more difficult to determine the partition size and scheduling parameters.

If jobs arrive in unpredictable bursts and the distribution of serial and parallel jobs varies over time, the partitioning of a cluster for parallel and serial jobs is particularly not desirable. Alternatively, the administrator of the cluster can try to configure a single scheduler to deal with both types of jobs as best as possible. Several scheduling systems exist which try to find a good compromise between serial and parallel jobs using techniques such as advanced reservation and backfilling [2]. These strategies are based on user estimates or predictive guesses of job execution times which are used to allow short jobs to run ahead of schedule on nodes which are reserved for a parallel job which is still waiting for a sufficient number of resources.

Both approaches (partitioning the cluster or balancing a single scheduler with backfilling) to mix parallel and serial jobs have the disadvantage that idle time is unavoidable if the spread of parallel and serial jobs varies over time or while a parallel job waits to get enough resources to run.

In this paper, we introduce an extension to the Sun Grid Engine [3] scheduler which allows the efficient mixing of parallel and serial jobs on a single cluster. Our approach is based on operating system virtualization techniques as provided by the Xen Hypervisor Monitor [4]. The resulting Xen Grid Engine (XGE) allows to balance parallel and serial jobs depending on user wishes in an easy and transparent manner. An added benefit of the proposal is the ability to checkpoint and migrate arbitrary serial jobs without requiring any modification of those jobs.

The paper is organized as follows. In section II, we introduce the problem posed by shared use clusters based on the MARC cluster running at the University of Marburg, Germany. In section III, we present our solution to the problem of serial and parallel job scheduling based on virtualization and present a proof-of-concept implementation. Section IV presents an evaluation of the solution based on results of tests and performance measurements done for an experimental testbed on the MARC cluster. Related work is discussed in section V. Section VI concludes the paper and outlines areas for future research.

II. Problem Statement

The motivation for the work presented in this paper originates from our observations of the daily utilization patterns of the MARC cluster located at the University of Marburg, Germany. The MARC cluster consists of 340 AMD Opteron CPU cores on 85 nodes (4 cores per node) shared between the departments of computer science, mathematics, physics and chemistry. The Sun Grid Engine (SGE) [3] scheduler is

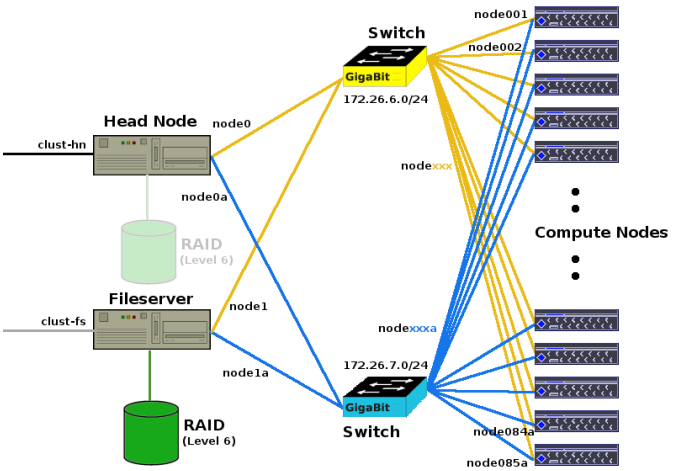

Fig. 1. Architecture of the MARC cluster [5]

Queue name	Time limit	Number of jobs completed
parallel	72 hours	9317
parallel_long	240 hours	237
serial	72 hours	38475
serial_long	240 hours	1149
serial_low	100 minutes	926212

TABLE I
OVERVIEW OF THE QUEUES AND THE COMPLETED JOBS ON THE MARC CLUSTER FROM JAN. 06 - AUG. 06

used to manage the cluster. Figure 1 shows the architecture of MARC.

In particular, two of the user groups have conflicting requirements. The physics department has a large number of short and long running serial jobs which they execute in a job farming pattern. The chemists have a large number of short running parallel jobs which they would like submit spontaneously to get a quick answer to research problems which crop up during their work.

Five queues exist which can be used to submit jobs, namely serial, serial_long, serial_low, parallel and parallel_long (see Table I). These differ in the CPU time granted for processes executed within these queues. Their time limit ranges from 100 minutes to 10 days. When submitting to a parallel queue, the user can specify how many CPU cores the job requires; 2, 4, 8, 16, 32, 64 or 128 are current legal values.

A parallel job can be submitted in one of two ways: (a) the job is submitted with the SGE *qsub* [6] option *-R* using reservation, (b) the job is submitted without the reservation option. These options only differ if not enough free resources exist to run the parallel job immediately. In the first case, all currently free cores are reserved for the execution of the parallel job. As soon as any jobs terminate, the corresponding CPU cores are reserved for the parallel job, up to the point where a sufficient number of resources has been reserved such that the parallel job can run. In the second case, without reservation, the parallel job waits in the job queue until a sufficient number of cores becomes idle by chance. If the cluster is operated under heavy load, this will probably never happen, which is why in practice all parallel jobs are submitted using reservation.

The drawback of simple reservation is that the nodes reserved for a parallel job run idle until the required free resources exist to run the parallel job, leading to a sub-optimal use of the cluster. A countermeasure to this waste of CPU time is to utilize backfilling. Algorithms such as EASY backfilling, where pending short jobs are allowed to move ahead, provided they do not delay the first job in the queue [7], or more conservative versions of backfilling, in which jobs move forward provided they do not delay any previously scheduled job [8], are used to fill idle time created when parallel and serial jobs are mixed. A lot of research has been poured into different backfilling strategies and their evaluation, see e.g. [9], [2], [10], [11], [12], [13], [14], [15], [16].

The number of strategies and configuration parameters creates a complex system of options which must be dealt with by the cluster administrator and the users. This leads to the problem that the selection and the exact configuration of a backfilling strategy can foster heated debates between the parallel and serial job submitters, since depending on the configuration of the scheduler either parallel or serial jobs will have a slight advantage. A further problem of backfilling is that as clusters grow and the number of jobs increases, the performance of a backfilling algorithm itself becomes an issue [17].

In the usage scenario shown in Table I, the number of serial jobs outweighs the number of parallel jobs. To maximize cluster efficiency, the cluster has been configured such that serial jobs have slight advantages in terms of waiting times, as shown in Table II. This is a thorn in the side of the chemists who say that they need the results of their short running parallel jobs in a timely manner (or it is not worth submitting them at all) and would like to see parallel jobs have shorter waiting times. In addition, the chemists would like to be able to utilize as many cores in parallel as possible so they can run their long running parallel jobs (normally run on 4 or 8 cores) on 128 or 256 cores as a short running parallel job, again in the interests of quick answers. The physicists counter saying that changing the scheduling policy would decrease the total cluster efficiency. Table III shows that most of the submitted programs are short running serial jobs that block most of the compute nodes. Since most parallel jobs are run on 4 or 8 cores, their runtimes can be significantly shortened by the allocation of 16, 32, 64, 128 or more CPU cores. This is currently impossible since the reservation limit has been set to 32 for performance reasons. Consequently, any job requesting more than 32 CPU cores will probably never run. The current waiting times for parallel and parallel long jobs depending on the number of CPU cores requested are shown in Table IV.

¿From the discussion above, the following requirements, divided into general requirements and requirements of the serial and parallel users (in *cursive*), can be derived:

1) User should be entitled to speedy job execution within their quotas.
2) Unused CPU time of a user may be consumed freely by

Queue name	Average wait time	Maximum wait time
parallel	111 min	12434 min
parallel_long	610 min	9199 min
serial	109 min	8176 min
serial_long	356 min	7100 min
serial_low	16 min	8726 min

TABLE II
TIME BETWEEN A JOB'S SUBMISSION AND START TIME.

Queue name	Number of jobs	Average run time
parallel	8922	489 min
parallel_long	225	6426 min
serial	35103	678 min
serial_long	1088	5828 min
serial_low	652033	26 min

TABLE III
RUNTIMES OF JOBS IN THE DIFFERENT QUEUES.

other users when needed.

3) *To maximize overall cluster performance, serial jobs should run whenever possible.*
4) *Parallel jobs should have waiting times as short as possible.*
5) *To minimize response time, parallel jobs should get as many CPUs as needed (definitively more than 32) without increasing the waiting time or reducing the overall cluster performance.*
6) Any modification of the scheduling strategy should be easy to use and transparent for administrators and users to avoid arguments.

III. VIRTUAL CLUSTER MANAGEMENT

In the following, we present the Xen Grid Engine (XGE) - our solution to satisfy the stated requirements of shared use cluster scheduling for parallel and serial jobs using virtualization technology. XGE is an extension of the Sun Grid Engine cluster management system based on the Xen Hypervisor Monitor [4].

The aim of XGE is to combine the advantages of cluster partitioning (simple but inefficient) and reservation with backfilling (complex customization, somewhat inflexible but more efficient). Using virtualization we intend to provide the ease of use of cluster partitioning while potentially beating the performance benefits of reservation with backfilling.

The basic approach is divided into three parts:

Number of nodes	Number of jobs	Average	Maximum
2	3334	166.2 min	3109 min
4	3141	52.4 min	12433.7 min
8	2743	75 min	5698.9 min
16	13	238.2 min	1503.6 min
32	10	222.3 min	723 min

TABLE IV
TIME BETWEEN SUBMISSION AND START OF PARALLEL JOBS DEPENDING ON THE NUMBERS OF CORES.

A. Virtual Partitioning

Instead of executing applications on the native Linux operating system, XGE uses the Xen hypervisor to separate every compute node of the cluster into a host platform and two virtual platforms. Following the Xen terminology, the host system is called dom0 whereas the virtual systems are called domU.

These domU instances are used for job processing, one for parallel and one for serial jobs. At any point in time, only one instance is running, depending on the mode the node is currently operating in, thus creating two virtual clusters, one for serial jobs and one for parallel jobs. Figure 2 shows the head node running SGE and XGE and one partitioned compute node. Currently, Xen is run in SMP mode, so each virtual node gets assigned all four cores of the node on which it is running. This, however, can easily be changed so that either two virtual nodes with two cores each, or four virtual nodes with one core each, are run in parallel on one physical node. The memory overhead is increased with each additional virtual node which runs in parallel, thus we currently assign all four cores to one virtual node. However, if a large number of jobs which do not work well with SMP and have only a small memory requirement, a significant performance gain is likely if four virtual nodes are used. This, however, is outside of the scope of this paper.

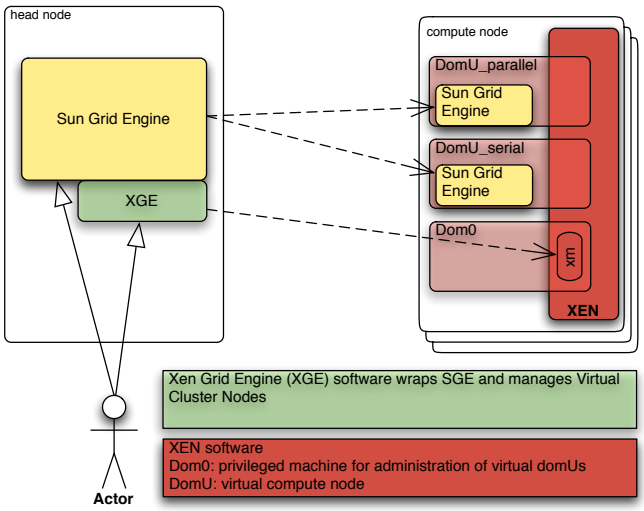

Fig. 2. XGE Architecture

Each of the virtual nodes is registered in at least one queue representing the kind of jobs. The serial domU should be registered within the serial queue(s), the parallel domU should registered within the parallel queue(s). Thus, every physical compute node is member of at least 2 queues, where at any point in time only the serial or parallel queues are reachable, since each physical node is only running one of the virtual machines. The Sun Grid Engine detects which nodes are reachable and will not schedule jobs on inaccessible nodes. The administrator of the cluster decides how many nodes activate the serial and how many activate the parallel virtual machines, thus partitioning the cluster to best suit the

users' current needs. In our case, the partitioning is based on the quotas of the serial and parallel users (which in turn in are based on the amount of money spent for the cluster acquisition.)

This solves requirements 1 and 4 stated in the previous section in the same way traditional partitioning of a cluster would, but reduces the overall performance of the cluster, since idle nodes in parallel mode will not be used by waiting serial jobs. Requirement 2 is only partially met, since only unused resources within the same group can be shared. Requirements 3 and 5 are not met outside of the quota based partitioning.

B. Dynamic Reassignment

However, since each physical node is equipped with two virtual machines, it is very simple to switch idle resources from one mode into the other in an on-demand fashion. If the nodes in a subset are overloaded and jobs must be queued, the idle nodes of the other subset can provide their computing power to that queue. The XGE automatically transfers nodes from the parallel virtual cluster into the serial virtual cluster if there are idle nodes available. The XGE does not transfer idle serial nodes (which are very rare in any case) to the parallel cluster. This will be explained in the next subsection. When the serial job is finished, the node is transferred back to the parallel cluster. This extension meets requirements 2 and 3 for serial jobs, since unused parallel idle time can be used up by waiting serial jobs. Two problems remain. First, if due to a transferral of resources, a newly submitted parallel job within its user's quota must wait for serial jobs to release those resources since not enough resources remain in the parallel cluster, requirement 2 would be broken. Second, requirement 2 is only met for serial jobs, since parallel jobs do not get idle resources from the serial cluster.

C. Transparent Job Suspension

The two remaining problems are solved by utilizing the functionality offered by Xen to suspend virtual machines while saving their state. In the case of a parallel job having to wait for a serial job to finish (a potentially long) execution on a freely given parallel resource, the XGE simply suspends the serial virtual machine and returns the parallel virtual machine to the parallel cluster. When the parallel queue is empty again, the serial job is automatically resumed. Any delays in the execution of these serial jobs is acceptable since they are working on free CPU cycles outside of their main quota.

The remaining problem that serial jobs can acquire idle parallel resources but parallel jobs can not acquire free serial resources stems from two issues : first, there are hardly any free serial resources and second, a node on which a parallel job is working cannot be reclaimed by the serial cluster, since parallel jobs cannot simply be hibernated by Xen because many parallel job managers do not recover from partial node failure and messages in transit would be lost. However, the following compromise has been reached: short running parallel jobs may acquire the nodes of long running serial jobs at any time within their quota. The long running serial job is suspended and the parallel job is executed on that node. On completion, the long running serial job is resumed. In our environment, this allows the chemists to submit highly parallel jobs (more than 128 cores) with short durations at any time as long as their quota is not exceeded, thus meeting requirement 5. Compared to the average runtime of serial long jobs, the delay caused by interrupting them by highly parallel short jobs is negligible.

To summarize, we have created an environment in which requirements 1 through 5 are met to the users' satisfaction. Users have timely access to all resources within their quota, serial jobs are free to use idle time from parallel gaps and parallel jobs are not restricted to 32 cores or long waiting times.

D. XGE Implementation

To satisfy requirement 6 ("ease of use for administrators"), the XGE is implemented as a number of add-on scripts and programs which can be deployed into existing SGE environments without the need to recompile or reinstall SGE. The administrator must specify the standard partition size and which queues can be suspended by the XGE for the virtual parallel cluster (in our case the serial long queue was used). Currently, the acquisition of idle parallel nodes by serial jobs is done on a simple FIFO basis. The selection of serial long jobs to be acquired by parallel jobs is currently also a list based approach. In both cases, a priority based approach will be part of future work to make the use and configuration of XGE easier for the administrator.

The users, on the other hand, have almost no added difficulty when using XGE. Within an unmodified SGE environment the job submission is separated into the following steps:

1) Job submission
   A job submitted by a user via the qsub command is forwarded to the master node.
2) Job scheduling
   The master node decides on which CPUs the job will be executed.
3) Job execution
   After the target CPU is fixed, the master node sends the job to the execution node and stores the appropriate accounting information in a database. Finally, the master nodes waits for a signal from the execution node, indicating the (successful) termination of the job.

Due to the transparent integration of XGE, users running serial jobs are not affected at all in their use of SGE. Parallel users must specify one additional command to the qsub program, specifying that the XGE engine should be used:

- –xge [submit job via the XGE and acquire needed resources from configured serial queues]

The original qsub program was replaced with a wrapper to pass common command line arguments directly to the SGE qsub program, whereas the –xge parameter starts the execution of applications provided by the XGE software.

Using XGE, the job submission is enhanced by additional steps, as shown in Fig 3:

1) A job is submitted with the replaced qsub command. The number of available nodes is determined.

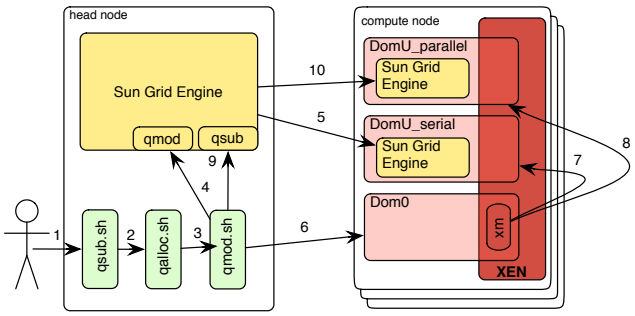

Fig. 3. Submission of a parallel job

2) The difference between the available and needed cores for the computation is allocated.
3) For each of the virtual machines to be suspended, the following tasks are applied:
4) XGE instructs the Sun Grid Engine to suspend the affected jobs.
5) The job is suspended within the SGE.
6) XGE looks up the responsible dom0 machine hosting the virtual Xen environment.
7) XGE suspends the virtual compute nodes responsible for the serial program execution.
8) XGE resumes the virtual machines for the upcoming parallel task.
9) XGE submits the job at the SGE
10) SGE executes the job on the new virtual machine(s).

After the job has finished, the set of suspended (virtual) nodes are reactivated and continue running their serial jobs.

The nodes which should be suspended can be limited to a queue. The ultimate decision which of the active nodes should be suspended to disk is made by the XGE software.

The communication between the submission node and a compute node's dom0 is based on *ssh*. If the XGE software needs to suspend or resume a virtual node, the software uses ssh to execute a command on the remote dom0 machine. Since the dom0 controls the domU instances on its node, a user with the appropriate permissions is able to start and stop the virtual machines.

This is achieved by the program *sudo* our software uses to communicate with the dom0 instances in the cluster. To authenticate on the remote machine, the software uses the public key method provided by ssh. For that, it is necessary that the privileged user's public key is stored at the remote machine's dom0 instance. All this is dealt with by the XGE software and is transparent to the users.

To summarize, with the XGE approach we are able to execute parallel and serial jobs without the need for fixed partitions or for reservation and backfilling strategies which cause unnecessary idle time in the cluster. Highly parallel jobs can be executed directly after submission while at the same time serial jobs can use up idle time of the entire cluster.

IV. Experiments and Results

The MARC cluster on which our work is based consists of 2 frontend nodes for job management and file services as well as 85 computing nodes. Each of these contain two dual core AMD Opteron 270 CPUs with 2.0 Ghz (i.e. 340 cores in total). 12 of the computing nodes are equipped with 16 GB of RAM and 73 with 8 GB of RAM. A brutto disk space of 4 TB is available and accessible from the compute nodes via NFS. A more detailed description of the hardware components can be found in [18]. Initial tests of XGE were conducted using 12 cores of this cluster.

While XGE reduces the overall idle time of the cluster by avoiding fixed partitioning and reservation, it introduces virtualization into cluster and thus a certain amount of overhead Four aspects must be studied: computational overhead, network I/O overhead, storage I/O overhead and the performance cost of storing and reviving virtual machines.

The computational overhead of Xen is very low compared to other virtualization solutions, being only a few percent slower that native Linux execution [4]. This has been independently confirmed by Clark et al. [19] and our users as well. In some scenarios, we even discovered that the execution of applications within a virtual environment is even faster than in a native environment. This has also been confirmed independently Gilbert et al. [20], but more work needs to be done to discover the reason for this result.

The network performance of Xen, however, is not quite as good. The maximum throughput achieved in a Xen environment is roughly 75 percent of the maximum throughput in a native Linux [21]. However, in the upcoming release of Xen it will be possible for the administrator to configure Xen in such a way that the domUs have direct access to the network hardware. This is totally acceptable for our scenario and will negate all performance issues.

The storage I/O performance using Xen varies depending on the application or on which benchmark is used, and on which file sizes the tests are done with. The I/O performance analysis of UnixBench[1] indicates that Xen is quite efficient. Bonnie[2] indicates that Xen is more efficient for small file sizes (200 MB and smaller) than for larger ones. We only had one report from our users about Xen difficulties during NFS file access. This, however, was a problem of the NFS configuration and not so much a Xen issue.

Finally, the overhead of suspending and resuming virtual cluster nodes needs to be evaluated. In Figure 4, the suspend and resume times are shown for different amounts of memory allocated to the virtual instances. The tests were conducted using up to 7 GB RAM within a virtual machine, resulting in an average of 150 seconds to suspend and 129 seconds to resume. The bottleneck in this case is the performance of the underlying storage system. In our case, we used 250 GB SATA hard disks [18]. The disk images (operating systems and suspend to disk) of the virtual machines are stored locally due to performance reasons.

In theory, it is also possible to save and restore virtual machines via NFS or a SAN backend. However, when we conducted tests with an easyRAID Q16+ SCSI-to-SATA storage subsystem with a capacity of 4 Terabytes, the repeated

[1] http://www.tux.org/pub/tux/niemi/unixbench/
[2] http://www.garloff.de/kurt/linux/bonnie/

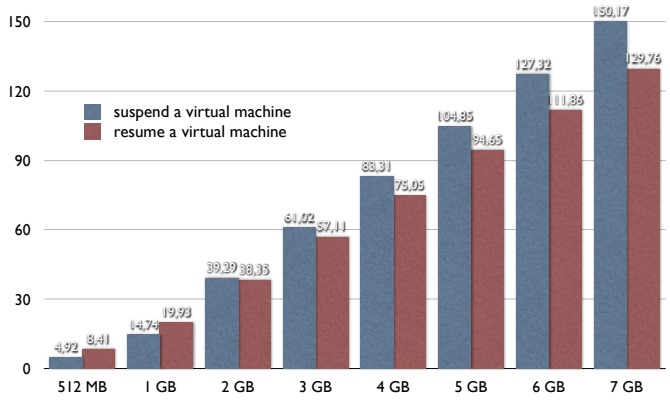

Fig. 4. Required time (in seconds) for virtual machine suspend and resume with different amounts of main memory

suspend and resume of virtual machines with up to 7 GB RAM resulted in a complete freeze of the NFS share. Due to the fact that the MARC cluster uses NFS to share data and the home directories between the execution nodes, a failure of the NFS server results in a complete cluster breakdown. However, since local storage space is not a problem, the need for remote storage is minimal and the problem can be avoided.

The different user groups working on MARC have experimented with the Xen environment using their production software and confirm the small performance cost of Xen. Based on that, they have come to the conclusion that the benefits of XGE outweigh their virtualization concerns and the test environment is due to be expanded into the production system soon.

V. Related Work

There is a lot of research in the area of virtualization, since it actually is a concept originating in the 1960s. Currently, Xen [4] is the open source favourite because of its near native performance and freely available source. In contrast to full virtualization, which is used in software like VMware [22], emulating a complete PC including the BIOS, Xen uses a technology called para-virtualization. Para-virtualization is a virtualization technique that presents a software interface to virtual machines that is similar but not identical to that of the underlying hardware and therefore requires the operating systems to be explicitly ported to run on top of the virtual machine monitor (VMM). Using para-vitualization, a higher performance can be achieved [19], [4].

There are several approaches which try to use the advantages of operating system virtualization in cluster or Grid environments, but not as an extension to a cluster scheduling system to improve coexistence of serial and parallel jobs.

Foster et al. [23] have identified the need to integrate the advantages of virtual machines in the cluster and Grid area. It is argued that virtual machines offer the ability to instantiate an independently configured guest environment for different users on a dynamic basis. In addition to providing convenience to users, this arrangement can also increase resource utilization since more flexible, but strongly enforceable sharing mechanisms can be put in place. The authors also identify that the ability to serialize and migrate the state of a virtual machine opens new opportunities for better load balancing and improved reliability that are not possible with traditional resources [23]. Virtual Workspaces [24] is a Globus Toolkit (GT4) based virtualization solution which allows Grid users to dynamically deploy and manage virtual machines in a Grid environment. To the best of our knowledge, the virtual workspaces approach does not interface with the scheduling system as XGE does, but operates on a higher level making a combination of virtual workspaces and XGE an interesting future prospect.

The Maestro virtual cluster [25] is a system for creating secure on-demand virtual clusters. The concept addresses on the fly virtual machine creation used in on-demand environment as well as the security advantages which the program execution within sandboxes brings.

VMPlant [26] is a Grid service for automated configuration and creation of virtual machines based on VMware which can be cloned and dynamically instantiated to provide homogenous execution environments within distributed Grid resources. The focus in this work is the definition of a framework for virtual machine management and the representation of software requirements through a direct acyclic graph.

Ghost Process [27] is a kernel service for process virtualization which allows process migration from one node to another for cluster balancing. This approach is based on a special Linux kernel and requires software to be developed using a so called GHOST-API. The proposal is not able to be implemented transparently without modifications of the underlying software or operating system.

VSched [28] is a system for distributed computing using virtual machines to mix batch and interactive VMs on the same hardware. Implemented as a user level program, it schedules virtual machines created by VMware GSX Server [29]. VSched is designed to execute processes within virtual machines during idle times on workstations. Processes are executed while users are not producing a high CPU load, e.g. while only using a word processor or surfing the web. Processes are executed with a lower CPU utilization (e.g. 10 percent) while the user is creating a high system load, using applications like Quake II [30]. This software is not intended to run on a cluster and probably not practicable in environments which usually have 100 percent CPU utilization.

None of the above approaches deal with the clashes between serial and parallel job execution in shared cluster environments.

VI. Conclusions

In this paper, we introduced a novel approach for cluster scheduling using operating system virtualization techniques. We presented a solution to enable the execution of short running parallel jobs within a cluster filled with serial jobs. Our implementation, a transparent extension to the Sun Grid

Engine, ensures fair use and enables maximum system performance by avoiding idle times used for reservation or static cluster partitioning. The solution does not require modifications of the Sun Grid Engine or the programs executed on the cluster.

Our experiments have shown that depending on the amount of memory used, the overhead of suspend and resume operations as well as XGE and SGE management overhead is negligible. The software has been tested and used on the MARC cluster of the University of Marburg, Germany, and is now being deployed into a production environment on a subset of the cluster nodes.

There are several areas for future work. Evaluating how XGE performs in a production environment will be the next step. Based on measurements and user feedback, different strategies for the acquisition of idle nodes and suspension of serial jobs should be investigated. Currently, Xen is run in SMP mode utilizing all 4 cores of a cluster node. A performance evolution of splitting each node into several serial virtual nodes could also be studied. Finally, the impact of Grid computing on job types, arrival times and management requirements is an interesting issue and would probably suggest an integration of our approach into existing Grid virtualization work such as [24] and [31].

REFERENCES

[1] D. Wright, "Cheap Cycles from the Desktop to the Dedicated Cluster: Combining Opportunistic and Dedicated Scheduling with Condor," in *Proceedings of the Linux Clusters: The HPC Revolution conference*, Champaign - Urbana, IL, June 2001.

[2] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Characterization of Backfilling Strategies for Parallel Job Scheduling," in *ICPPW '02: Proceedings of the 2002 International Conference on Parallel Processing Workshops*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 514–522.

[3] Sun Gridengine Developers, "Sun GridEngine Website," 2006, http://gridengine.sunsource.net.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM Press, 2003, pp. 164–177.

[5] T. Gebhardt, "Configuration of the MARC Cluster," 2006, http://www.uni-marburg.de/hrz/infrastruktur/zserv/cluster/config.

[6] "Manpage of the qsub Program," 2006, http://gridengine.sunsource.net/nonav/source/browse/~checkout~/gridengine/doc/htmlman/htmlman1/qsub.html.

[7] D. A. Lifka, "The ANL/IBM SP Scheduling System." in *Job Scheduling Strategies for Parallel Processing (JSSPP)*, 1995, pp. 295–303.

[8] A. M. Weil and D. G. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling." *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529–543, 2001.

[9] B. G. Lawson and E. Smirni, "Multiple-Queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems." in *Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2002, pp. 72–87.

[10] D. Zotkin and P. J. Keleher, "Job-Length Estimation and Performance in Backfilling Schedulers," in *HPDC '99: Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 1999, p. 39.

[11] B. Lawson and E. Smirni, "Self-Adaptive Scheduler Parameterization via Online Simulation," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers*. Washington, DC, USA: IEEE Computer Society, 2005, p. 29.1.

[12] S.-H. Chiang and F. Chuyong, "Re-evaluating Reservation Policies for Backfill Scheduling on Parallel Systems," *16th IASTED Int'l Conf. on Parallel and Distributed Computing and Systems (PDCS)*, 2004.

[13] P. J. Keleher, D. Zotkin, and D. Perkovic, "Attacking the Bottlenecks of Backfilling Schedulers," *Cluster Computing*, vol. 3, no. 4, pp. 245–254, 2000.

[14] W. Smith, I. T. Foster, and V. E. Taylor, "Predicting Application Run Times Using Historical Information," in *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. London, UK: Springer-Verlag, 1998, pp. 122–142.

[15] B. G. Lawson and E. Smirni, "Multiple-Queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems," *SIGMETRICS Performance Evaluation Review*, vol. 29, no. 4, pp. 40–47, 2002.

[16] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling Using Runtime Predictions Rather than User Estimates," School of Computer Science and Engineering, Hebrew University of Jerusalem, Tech. Rep. TR 2005-5, 2003.

[17] A. Haas, "Specification of the Resource Reservation and Backfilling Grid Engine 6.0 scheduler enhancement," 2004, http://gridengine.sunsource.net/nonav/source/browse/~checkout~/gridengine/doc/devel/rfe/resource_reservation.txt".

[18] Thomas Gebhardt, "Description of the MARC Cluster Hadrware," 2006, http://www.uni-marburg.de/hrz/infrastruktur/zserv/cluster/hardware.

[19] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J. N. Matthews, "Xen and the Art of Repeated Research." in *USENIX Annual Technical Conference, FREENIX Track*, 2004, pp. 135–144.

[20] L. Gilbert, J. Tseng, R. Newman, S. Iqbal, R. Pepper, O. Celebioglu, J. Hsieh, and M. Cobban, "Performance Implications of Virtualization and Hyper-Threading on High Energy Physics Applications in a Grid Environment," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers*. Washington, DC, USA: IEEE Computer Society, 2005, p. 32.1.

[21] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel, "Diagnosing Performance Overheads in the Xen Virtual Machine Environment," in *VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*. New York, NY, USA: ACM Press, 2005, pp. 13–23.

[22] VMWare INC, "VMWare Homepage," 2006, http://www.vmware.com.

[23] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang, "Virtual Clusters for Grid Communities," in *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 513–520.

[24] The Globus Project, "Virtual Workspaces," 2006, http://workspace.globus.org/.

[25] N. Kiyanclar, G. A. Koenig, and W. Yurcik, "Maestro-VC: A Paravirtualized Execution Environment for Secure On-Demand Cluster Computing," in *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*. Washington, DC, USA: IEEE Computer Society, 2006, p. 28.

[26] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo, "VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing," in *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2004, p. 7.

[27] G. Valle, R. Lottiaux, D. Margery, C. Morin, and J.-Y. Berthou, "Ghost Process: a Sound Basis to Implement Process Duplication, Migration and Checkpoint/Restart in Linux Clusters," in *The 4th International Symposium on Parallel and Distributed Computing*, Lille, France, July 2005, pp. 97–104.

[28] B. Lin and P. A. Dinda, "VSched: Mixing Batch And Interactive Virtual Machines Using Periodic Real-time Scheduling," in *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2005, p. 8.

[29] VMWare INC, "VMWare GSX Server," 2006, http://www.vmware.com/products/server/.

[30] Id Software, Inc., "Quake II Homepage," 2006, http://www.idsoftware.com/games/quake/quake2/.

[31] M. Smith, T. Friese, M. Engel, and B. Freisleben, "Countering Security Threats in Service-Oriented On-Demand Grid Computing Using Sandboxing and Trusted Computing Techniques," *Journal of Parallel and Distributed Computing*, vol. 66, no. 9, pp. 1189–1204, 2006.