Extending The Linux Operating System For Grid Computing

Nur Hussein
Postgraduate Lab,
School of Computer Science,
Universiti Sains Malaysia,
11800 Pulau Pinang, Malaysia.
hussein@cs.usm.my

Constantine Kolivas
Australian & New Zealand
College of Anaesthetists,
630 St Kilda Road, Melbourne
Vic 3004, Australia.
kernel@kolivas.org

Fazilah Haron,
Chan Huah Yong
Grid Computing Lab,
School of Computer Science,
Universiti Sains Malaysia,
11800 Pulau Pinang, Malaysia.
fazilah@cs.usm.my
hychan@cs.usm.my

ABSTRACT
The emergence of metacomputing and grid technologies as a new trend in high performance computing has opened up many interesting problems in the implementation of resource-sharing and efficient scheduling of compute-intensive processes. While much work has been done to build grid services on top of existing infrastructure, little has been done to make the underlying operating system aware of these new applications. If extensions for facilitating grid computing could be built into the operating system kernel, it would simplify the job of grid service implementers in userspace for they could rely on the transparent services of the kernel for tasks such as process checkpointing and resource discovery. This paper discusses the implementation of grid-friendly extensions to the Linux operating system; process checkpointing, userspace filesystems and a fair-scheduling algorithm optimized for compute-intensive applications. The fair-scheduling algorithm is benchmarked and compared to existing Linux schedulers, and the preliminary results show an increase in system throughput with the modified scheduling algorithm.

Categories and Subject Descriptors
C.2.4 [**Computer-Communications Networks**]: Distributed Systems - *Network operating systems*; D.4.8 [**Operating Systems**]: Performance – *measurements, operational analysis*

General Terms
Algorithms, Measurement, Performance, Design, Experimentation.

Keywords
Linux, grid computing, checkpointing, distributed file systems, fair scheduling.

Asia Pacific Advanced Network/QUESTnet 2004, 2-7 July 2004, Cairns, Australia.
Network Research Workshop 2004, 2-7 July 2004, Cairns, Australia.

1. INTRODUCTION

Grid computing [1] is an emerging new paradigm of distributed computing which allows the sharing and aggregation of distributed resources in a unified manner. There are numerous software projects such as Globus [2] and Condor [5] which help enable this resource-sharing in an organized and systematic way. However, the underlying operating systems that are used to implement grid computing are usually unaware of this resource sharing. Therefore, if the operating system were built with certain extensions to take into account this distributed nature of resource sharing, it would make implementation of certain grid services simpler and more transparent. We have added several modifications to the Linux operating system kernel to complement the implementation of higher-level grid services in userspace.

Linux was chosen as the platform of choice due to its emerging popularity as a scientific and grid computing platform. The modifications added are the aggregation of several Linux kernel projects that was integrated into a customized Linux kernel which provides the following:

- Process checkpointing
- Extends the VFS [4] interface for customized filesystems implementable in userspace
- Enables fair scheduling in the Linux kernel

These modifications help userspace grid programs execute in a more flexible environment. This paper will elaborate on each of these different modifications and the rationale behind the inclusion of each.

1.1 Process Checkpointing

Grid computing usually involves the execution of long-running computational processes. Applications such as molecular modeling, DNA and protein analysis, mathematical algorithms and other scientific applications which are the target usage of grid technologies usually run for hours or days, even on modern hardware. Therefore, it would be advantageous if the execution state of these applications could be captured and saved to disk, either for backup purposes (so that work won't be lost if there is

node failure) or for process migration (process state can be transferred to another machine and restarted there).

Some operating systems provide a mechanism for process checkpointing that enables applications to be suspended and restored. However, this functionality is still missing in the official Linux kernel. Checkpointing in our custom Linux kernel is achieved through the addition of the EPCKPT checkpoint code, which is work initially done by Pinheiro [8]. With this additional code, it is possible to implement resource schedulers that have the freedom to transparently checkpoint, and if required, migrate processes in the distributed system. Since this mechanism is implemented in kernelspace, checkpoint and restart of the processes can be done almost completely transparently.

1.2 Userspace Filesystem Extensions

One of the philosophies of the Unix operating system is that "everything is a file". It was this concept that was carried over into the design of the Plan 9 Operating System from Bell Labs [7], where distributed resources are also represented in the filesystem. This concept was the inspiration for adding Malita's Linux Userspace Filesystem (LUFS) [6] into the custom kernel. LUFS augments VFS with a generic filesystem interface that "exports" POSIX system calls such open and read into userspace. Therefore, it is possible to implement VFS functions in userspace, allowing grid service implementers to transparently handle filesystem calls for distributed processes without having to resort to using a global filesystem for every single participating computing node. It also allows complex filesystem-based modules to be added to the operating system without the overhead of bloating the kernel proper, since the code will run in userspace.

One such module is the distributed /proc interface, which allows the representation of remote resources inside the hierarchy of the filesystem. Traditionally, the /proc interface contains information on memory, processor availability and other system information. The distributed /proc interface contains information about remote computing nodes and the information about each resource in the remote nodes. Like Plan 9, this information is not polled, but generated (queried from the remote nodes) when the representative distributed /proc entry is queried by any process. Therefore, this simplifies the implementation of a resource scheduler for the distributed system, as it does not need to use any special API for resource discovery as normal POSIX system calls can be used. In Figure 1, there are n nodes visible to a particular node. Besides the local /proc filesystem, the remote nodes' /proc filesystems are also accessible to the local node because they are mounted via userspace filesystem modules. Any process can transparently access (query) the remote /proc filesystems as if they were on the local node. A userspace filesystem will handle all accesses to these special files, and send network queries to the remote nodes.

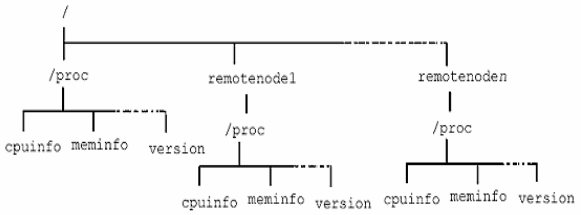

Figure 1 : VFS hierarchy showing mounted remote /proc filesystems

1.3 Fair Scheduling In Linux

The official Linux kernel was designed to be as generic as possible, to support all kinds of architectures from high-end mainframes to handheld computers. It has to perform well on servers and desktops so the process scheduler tries to adapt to the different workloads that it might encounter. Desktop users in particular want a graphical user interface that is responsive and smooth. To achieve this, the scheduler has to unfairly favor the processes that are interactive, such as processes that use or manage the GUI. However, in certain computational environments such as a compute-server farm, it is not necessary to perform such scheduler optimizations on interactive processes, as most servers are not used as desktops. Therefore, for the purpose of compute-intensive servers, Kolivas [3] devised a simpler fair-scheduling algorithm. This work is based on Molnar's O(1) scheduler which is more scalable than the default 2.4 kernel scheduler and is now part of the new 2.6 kernel. This algorithm works as follows:

1. There are two queues of processes, one for active processes and another for expired processes. Each queue is effectively sorted by priority.

2. Every process of the same priority X is given N units of timeslice

3. If a process has priority Y and Y > X then it is given M units of timeslice where M > N

4. Unless preempted by another process of a higher priority (if one wakes up), a process will run until it's timeslice expires or it goes to sleep voluntarily

5. When a process's timeslice expires, it is removed from the active queue and inserted into the expired queue. Once the active queue becomes empty, the expired queue becomes the active queue and the active queue becomes the expired queue.

In the official Linux 2.4 and 2.6 kernels, the priority value of the default scheduler is a function of its "nice" value and its sleep behavior. This priority value, called the dynamic priority of a process is the effective priority given to tasks after an interactivity estimation algorithm takes into account how much the process needs to respond to user input/output. With the fair-scheduling algorithm, we allow processes of the same "nice" level to not ever have dynamic priorities higher than processes with an equal "nice" value.

Therefore, unlike the default scheduler, there is never preemption by tasks of equal "nice" value. Hence, if all processes run at the same "nice" level they will always run using up a full timeslice. When preemption does occur because of a higher priority task, it is delayed for a short time so that even lower priority tasks get to run for some time bound to the CPU before being expired. This effectively increases the throughput of the operating system, because the longer a task is bound to the CPU the more it benefits from the CPU's cache. Also, the default timeslices for this modified scheduler have been increased (see Table 1). We will compare the results of a throughput test between kernels in Section 2, where we see a notable increase in both throughput and scalability of the modified Linux kernel.

In a grid computing environment, it is desirable for the fair scheduling of processes, for the benefit of the added throughput, and also to ensure that the priorities assigned to the processes by the user have maximum effect on the way the computer schedules them. This is because a distributed system such as the grid will have many different users, with different levels of authority. Enforcing the priority levels as fairly as possible ensures users of different priorities get their appropriate percentage of CPU time.

Table 1 : Scheduler timeslices comparison between default kernel 2.4.22 and modified kernel 2.4.22

Nice Value	Default Linux 2.4.22 timeslices (milliseconds)	Modified Linux 2.4.22 timeslices (milliseconds)
-20	110	300
0	60	150
20	10	10

2. PERFORMANCE EVALUATION OF THE MODIFIED KERNEL

We have carried out several tests of the modified Linux kernel to compare its performance against the default kernel. In particular, we wanted to see the effect of the fair scheduling policy on the overall throughput and scalability of the kernel. The tests were carried out with the help of Open Source Development Lab's STP program. The tests were run on an 8-way 700 Mhz Pentium III Xeon with 1024K of Level 2 cache on each of the eight processors. The total system RAM for the machine is 8 GB, and the 40 hard disks run on SCSI-RAID controllers. The test suite run was Reaim [9], which is a modernized version of AIM technologies AIM 7 and AIM 9 benchmarks. It runs an increasing number of processes that do a number of tas ks to completion, and measures the time taken. Three kernels versions were tested; the standard Linux 2.4.22, the standard Linux 2.6.0-test9 and the modified kernel 2.4.22 that contains the fair scheduler. The graphs below show the number of jobs completed per minute against the number of processes that were created to run on the system.

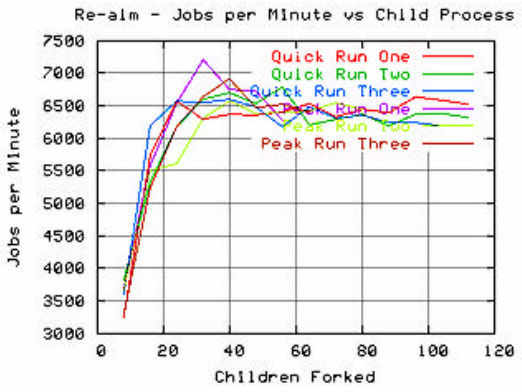

Figure 2 : Reaim results for Linux kernel 2.4.22

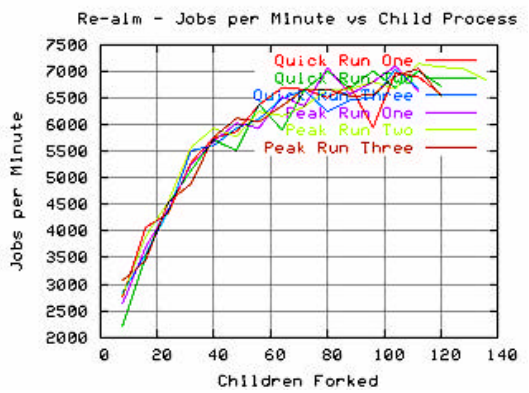

Figure 3 : Reaim results for Linux kernel 2.6.0-test9

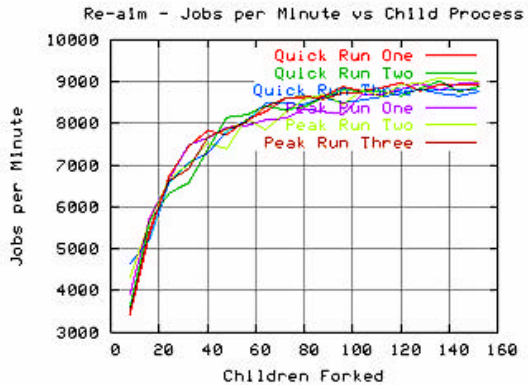

Figure 4 : Reaim results modified Linux kernel 2.4.22 with fair scheduler.

As can be seen from the results, the fair scheduler can handle a peak of approximately 9000 jobs a minute, compared to 6500 jobs a minute and 7000 jobs a minute for the default schedulers in kernel 2.4.22 and the new kernel 2.6.0-test9. This is an increase in throughput of approximately 40% compared to the default scheduler in an equivalent Linux 2.4.22 kernel. This is the effect of processes being able to take advantage of the CPU cache each time it is scheduled, since each process gets more CPU time to do work compared to the default kernel.

3. CONCLUSION AND FUTURE WORK

In this paper we have described the addition of kernel components that make the operating system a more flexible, transparent and feature-rich environment for the development of grid services. The addition of kernel checkpointing, customizable userspace filesystems and more computational-centric scheduling have provided the groundwork for more complex mechanisms to built on top of it. A process migration facility can be built on top of this foundation, and a custom userspace filesystem module is required to handle distributed file I/O for migrated processes. Currently, the checkpointing mechanism has several limitations that are inherited from the implementation of EPCKPT. This includes the inability to checkpoint sockets and certain System V objects such as shared memory and semaphores are yet unsupported. However, such additions are implementable and are being worked on. A

slightly more complicated problem involves the checkpointing of processes that access certain physical devices, as the Linux driver model is not built to be compatible with checkpointing.

The addition of the fair scheduler with longer timeslice quanta showed an improvement in total system throughput. Because of this, it is more efficient for servers running compute-intensive processes. However, there is a side effect to this if the computer running this modified kernel is used as a desktop. The longer timeslices given to processes result in a slightly sluggish GUI desktop, which means this kernel is only suitable to be deployed in servers that are not primarily used with attached GUI terminals. This is usually not a problem in clusters and compute-server farms that usually run "headless" (without a terminal or console attached) anyway.

4. REFERENCES

[1] Foster, I. and Kesselman, K., "The Grid: Blueprint for a New Computing Infrastructure", Morgan-Kaufman Publishing, San Francisco, 1999.

[2] Foster, I. and Kesselman, K., "Globus: A Metacomputing Infrastructure Toolkit", Proceedings of the Workshop on Environments and Tools for Parallel Scientific Computing, SIAM, Lyon, France, August 1996, pp. 115-128.

[3] Kolivas, C., "The LCK Patchset", <http://www.plumlocosoft.com/kernel/>, May 2004.

[4] Kleiman, S.R., "Vnodes: An Architecture for Multiple File System Types in Sun Unix", USENIX Association: Summer Conference Proceedings, Atlanta, 1986, pp. 238-247.

[5] Litzkow, M., Livny, M., and Mutka M., "Condor : A hunter of idle workstations", Proceedings of the Eighth Conference on Distributed Computing Systems, San Jose, California, June 1988.

[6] Malita, F.,"Linux Userspace Filesystem", <http://lufs.sourceforge.net/lufs/ >, November 2003.

[7] Pike, R., Presotto, D., Thompson, K. and Trickey, H. "Plan 9 from Bell Labs", Proc. of the Summer 1990 UKUUG Conf., London, July 1990, pp. 1-9

[8] Pinhiero, E., "Truly-Transparent Checkpointing of Parallel Application", Working Draft. <http://www.research.rutgers.edu/~edpin/epckpt/>, November 2003.

[9] White, C., "Performance Testing The Linux Kernel : The Reaim Workload", Proceedings of the Linux Symposium, Ottawa, Ontario, Canada, 2003, pp481-494