A Summary of Research in System Software and Concurrency at the University of Malta: I/O and Communication

Joseph Cordina

Department of Computer Science and AI,
University of Malta

Abstract. Traditional operating systems and commodity hardware are never used to their full potential due to underlying design limitations. Applications that make use of blocking system calls incur large overheads on the operating systems and in turn end up wasting CPU resources. In addition, traditional solutions are not adequate for high-performance networking. In this report, we present a summary of the research conducted by the System Software Research Group (SSRG) at the University of Malta. We discuss some of the solutions we have developed and pinpoint their effectiveness to solve each of the above problems.

1 Introduction

Facilities commonly offered by operating systems, such as memory protection and I/O resource usage, often result in huge bottlenecks to applications demanding huge computational resources. The traditional way of accessing resources is through system calls. Each system call demands switching of protection boundaries which is quite computationally expensive. In addition, CPU time and consequently application time is wasted when waiting for these resources to terminate their requests. Such slow resources are disk usage, network communication and even memory access.

The System Software Research Group[15] (SSRG) was set up in the Department of Computer Science at around 1999. Its main aim is to conduct research in the area of System Software. The above problems proved to be an ideal direction for further research. In [22] Vella proposes several methods that the SSRG has worked on to improve performance at the application level. Some of these optimisations are based on user-level multithreading techniques. Unfortunately user-level thread schedulers fail in their utility when invoking system calls, in that the user-level thread scheduler becomes blocked until the underlying kernel thread itself is re-scheduled by the kernel. The propose several solutions to this problem developed within the research group and analyse their performance.

Computing today is tightly dependent on network performance. In tests performed in [6] we have shown the limitations proposed by current commodity hardware in dealing with high bandwidth networking. Thus this research group has also concentrated on removing the bandwidth bottlenecks while still using commodity hardware. We show that cheap hardware solutions are able to cater for highly demanding network applications.

2 Operating System Integration

Any application that makes use of system resources such as disk I/O and networking needs to make calls into the underlying kernel. The reason for this barrier is due to current operating systems that

protect the above application from access intricacies and to protect one application from another. Unfortunately when accessing slow resources, the application remains blocked until the resource can service the request. In the mean time, other kernel processes or threads can continue executing.

In highly demanding applications that make use of such resources, the common alternative is to make use of multiple kernel entities that will be utilised whenever the running thread gets blocked. This provides better CPU usage. Yet this is not ideal due to the expense of creating new kernel threads and the expense of switching to other kernel threads. In addition, when utilising user-level thread schedulers, when one user level thread makes a blocking call, all the other user level threads are not able to continue executing. Within the SSRG we have investigated various solutions to this problem. One of the most basic solutions is the use of *wrappers*. This is a modification of potentially blocking system calls, such that such calls invoke a second kernel thread that continues to execute other user-level threads that are not blocked. Whenever the blocked user-level thread unblocks, the second kernel thread halts and the original user-level thread continues executing. Wrappers provide an adequate solution when the system call actually blocks. Unfortunately system calls that do not block (for example when a *read* call is issued that already has data locally in a memory buffer) cause unnecessary overheads. In addition, wrappers provide a solution which is not transparent to the application programmer.

In addition to system calls, the application may also block whenever accessing memory pages that are not readily available (memory pages have to be initialised or have been swapped out to disk space). Such page faults still cause the application to block for an undefined amount of time, wasting CPU time. Wrappers are obviously not capable of solving such problems. Within the SSRG we have concentrated on providing solutions that achieve high performance while being transparent to the programmer.

2.1 Activations

We have developed a solution that is capable of solving the above problems. Through Linux kernel modifications, we were able to use a mechanism such that whenever a system call blocks, a call is made from the underlying kernel to the blocked application. This kind of information allows the application to react accordingly to the changing environment of the operating system. By making use of thread pools within the kernel and utilising user-level thread schedulers, we were able to solve most of the problems associated with blocking calls. Our results[3] have shown that this solution is able to achieve high performance when applied to an application that is dependent on a large number of potentially blocking system calls such as a web server. This solution was also extended to be applied to SMP architectures. Unfortunately due to limitations in the underlying kernel modifications, this solution cannot be applied to blocking due to page faults.

2.2 Shared Memory Asynchronous Communication

In comparison to the above solution, we have developed a further mechanism whereby a shared memory region is used by the application and the underlying kernel to pass information to each other. This mechanism offers a solution scalable primitive that was applied to the above blocking call problem and also to the extended spinning problem[1][18]. By making use of this asynchronous mechanism, the kernel can create new kernel threads whenever a blocking system call occurs and can also inform the user-level thread scheduler whenever a system call unblocks. This solution was found to be comparable in performance to the activation mechanism while providing additional scalability.

[1] This is a problem that occurs whenever several processes make use of a shared lock to execute critical sections in a multiprogrammed environment[19]

3 Network Communication

Most applications today depend heavily on network connectivity and fast bandwidth. While CPU speed is ever increasing, advances in network technologies, interconnecting architectures and operating system software has not kept pace. The SSRG has performed several tests to analyse bottlenecks within the connection pipeline. We have found that that on commodity platforms the main bottlenecks are the operating system network stack, common interconnecting architectures such as the PCI and the network protocol itself. We set out to prove that gigabit connectivity can be achieved on commodity platforms and Ethernet standards.

3.1 Ethernet Bandwidth

Various tests were carried out by Dobinson et al.[2][13] to analyse network traffic on Ethernet hardware. The test controller used was the Alteon Tigon controller[16], a gigabit Ethernet controller that allows custom software to be uploaded and executed on board the card. It was found out that making use of larger Ethernet packets[3] it was possible to achieve gigabit data bandwidth on the physical line.

The SSRG then concentrated on solving the problem in other layers of the communication pipeline. In his project, Wadge[24] made use of the processing power of the Alteon card by modifying the firmware such that bottlenecks on the PCI were bypassed for Ethernet packets. This was performed by transferring large chunks of data through the PCI instead of the traditional 1.5K packets. When using Ethernet packets, we achieved almost gigabit bandwidth, a first when making use of 33MHz, 32 bit PCI. In addition, the group took ideas from MESH[2], a user-level thread scheduler integrated with Ethernet communication, to make use of an area of memory accessible only to the application thus allowing user-level access to the Alteon's frame buffer.

3.2 TCP/IP bandwidth

The group still desired to extend the above results further to bypass restrictions found in most networking stacks of traditional operating systems. It was found that the cost of crossing the protection barrier between the application and the underlying kernel, coupled with several data copies that are made within the kernel for TCP/IP connection, severely degrades the point to point bandwidth. In addition, we found that CPU utilisation is substantial for high bandwidth networking to the point that on commodity hardware, applications are devoid of CPU resources. Thus we have investigated the development of user-level networking, where the kernel does not take part in the communication process[6]. This was achieved through the use of the Alteon Ethernet Controller and an area of memory reserved to the application. While this mechanism does not offer the traditional protection facilities provided by the operating system, we felt justified in bypassing this restriction due to the high-demand environment of this application. We have also developed a TCP/IP stack, solely at the user level and using zero-copy communication. We managed to achieve very high bandwidth rates with very low CPU utilisation on relatively slow commodity hardware, showing that demanding applications *can* be developed and executed on inexpensive hardware.

[2] In collaboration with CERN and the SSRG

[3] Known as Jumbo packets that are 9K large instead of the standard 1.5K packets

3.3 User-Level Threads and Networking

To make use of the high-performance techniques proposed in [22] and above, the research group has developed certain extensions to SMASH[8] (a user-level thread scheduler). SCOMM[4] is a product that allows CSP-like channel connectivity on the network. It makes use of TCP/IP yet was developed to allow integration of other network drivers underneath. It solves the blocking problem of traditional communication calls by allocating the responsibility of network communication to another kernel thread, thus avoiding the user-level thread scheduler from blocking. In addition Nickovic[17] has further enhanced SCOMM by designing and implementing shared variables though this communication protocol. This mechanism was built without requiring any central servers for variable consistency.

4 Conclusion

We have shown the various areas that have tackled by the System Software Research Group in terms of I/O and communication. We solved the problem of applications that make use of blocking system calls and applied our solutions to maximise CPU usage. In addition, we have developed several high-performance networking solutions that cater for high bandwidth and low CPU usage. While various projects have been developed from this research, and most of the work is at par or better with the best research done in this area, there is still more work that we plan to achieve. We plan to integrate our user-level TCP/IP package with SMASH. When integrated, this would provide a high-performance solution towards computation with high-bandwidth networking at very little cost. In addition we plan to work towards building higher level applications on top of SMASH. We aim to be able to apply peer to peer algorithms[14] and distributed data structures[7] to provide a framework for high-performance distributed computation and space storage.

The System Software Research, while burdened by limited resources, both in terms of hardware and man power, has managed to achieve products of very high quality. We have developed several novel solutions that are capable of servicing a large number of demanding request making full use of the capabilities of the underlying hardware.

References

1. S. Abela. Improving fine-grained multithreading performance through object-affinity scheduling. B.Sc. I.T. Final Year Project, Department of Computer Science and Artificial Intelligence, University of Malta, June 2002.
2. M. Boosten. *Fine-Grain Parallel Processing on a Commodity Platform: a Solution for the ATLAS Second Level Trigger*. PhD thesis, Eindhoven University of Technology, 1999.
3. A. Borg. Avoiding blocking system calls in a user-level thread scheduler for shared memory multiprocessors. B.Sc. I.T. Final Year Project, Department of Computer Science and Artificial Intelligence, University of Malta, June 2001.
4. S. Busuttil. Integrating fast network communication with a user-level thread scheduler. B.Sc. I.T. Final Year Project, Department of Computer Science and Artificial Intelligence, University of Malta, June 2002.
5. J. Cordina. Fast multithreading on shared memory multiprocessors. B.Sc. I.T. Final Year Project, Department of Computer Science and Artificial Intelligence, University of Malta, June 2000.
6. J. Cordina. High performance TCP/IP for multi-threaded servers. Master's thesis, University of Malta, March 2002.
7. J. Cutajar. A scaleable and distributed B-Tree with parallel out-of-order traversal. B.Sc. I.T. Final Year Project, Department of Computer Science and Artificial Intelligence, University of Malta, June 2003.

8. K. Debattista. High performance thread scheduling on shared memory multiprocessors. Master's thesis, University of Malta, February 2001.
9. K. Debattista and K. Vella. High performance wait-free thread scheduling on shared memory multiprocessors. In H.R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 1022–1028, June 2002.
10. K. Debattista, K. Vella, and J. Cordina. Cache-affinity scheduling for fine grain multithreading. In J.S. Pascoe, P.H. Welch, R.J. Loader, and V.S. Sunderam, editors, *Proceedings of Communicating Process Architectures 2002*, volume 60 of *Concurrent Systems Engineering*, pages 135–146. IOS Press, September 2002.
11. K. Debattista, K. Vella, and J. Cordina. Wait-free cache-affinity thread scheduling. *IEE Proceedings - Software*, 150(2):137–146, April 2003.
12. R.W. Dobinson, E. Knezo, M.J. LeVine, B. Martin, C. Meirosu, F. Saka, and K. Vella. Characterizing Ethernet switches for use in ATLAS trigger/DAQ / Modeling the ATLAS second level trigger Ethernet network using parameterized switches. In *Proceedings of the IEEE Nuclear Science Symposium: Workshop on Network-Based Data Acquisition and Event-Building*, October 2000.
13. R.W. Dobinson, E. Knezo, M.J. LeVine, B. Martin, C. Meirosu, F. Saka, and K. Vella. Testing and modeling Ethernet switches for use in ATLAS high level triggers. *IEEE Transactions on Nuclear Science*, 48(3):607–612, June 2001.
14. J. Farrugia. P2P .NET-a peer to peer platform for the Microsoft .NET framework. B.Sc. I.T. Final Year Project, Department of Computer Science and Artificial Intelligence, University of Malta, June 2003.
15. K. Vella J. Cordina, K. Debattista. System software research group website. URL:`http://cs.um.edu.mt/~ssrg`.
16. Alteon Networks. Gigabit ethernet/PCI network interface, host/NIC software interface definition. *Alteon Networks Documentation*, June 1999.
17. D. Nickovic. Distributed shared variables for user-level fine grained multithreading. B.Sc. I.T. Final Year Project, Department of Computer Science and Artificial Intelligence, University of Malta, June 2003.
18. I. Sant. An asynchronous interaction mechanism between the kernel and the user application. B.Sc. I.T. Final Year Project, Department of Computer Science and Artificial Intelligence, University of Malta, June 2003.
19. A. Tucker and A. Gupta. Process control and scheduling issues for shared-memory multiprocessors. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 159–166, 1989.
20. K. Vella. CSP/occam on networks of workstations. In C.R. Jesshope and A.V. Shafarenko, editors, *Proceedings of UK Parallel '96: The BCS Parallel Processing Specialist Group Annual Conference*, pages 70–91. Springer-Verlag, July 1996.
21. K. Vella. *Seamless Parallel Computing on Heterogeneous Networks of Multiprocessor Workstations*. PhD thesis, University of Kent at Canterbury, December 1998.
22. K. Vella. A summary of research in system software and concurrency at the University of Malta: multithreading. In *Proceedings of the Computer Science Annual Research 2003, University of Malta*, June 2003.
23. K. Vella and P.H. Welch. CSP/occam on shared memory multiprocessor workstations. In B.M. Cook, editor, *Proceedings of WoTUG-22: Architectures, Languages and Techniques for Concurrent Systems*, volume 57 of *Concurrent Systems Engineering*, pages 87–120. IOS Press, April 1999.
24. W. Wadge. Achieving gigabit performance on programmable ethernet network interface cards. B.Sc. I.T. Final Year Project, Department of Computer Science and Artificial Intelligence, University of Malta, June 2001.