Balancing Computational Science and Computer Science Research on a Terascale Computing Facility

Calvin J. Ribbens, Srinidhi Varadarjan,
Malar Chinnusamy, and Gautam Swaminathan

Department of Computer Science, Virginia Tech,
Blacksburg, VA 24061
{ribbens, srinidhi, mchinnus, gswamina}@vt.edu

Abstract. The design and deployment of Virginia Tech's terascale computing cluster is described. The goal of this project is to demonstrate that world-class on-campus supercomputing is possible and affordable, and to explore the resulting benefits for an academic community consisting of both computational scientists and computer science researchers and students. Computer science research in high performance computing systems benefits significantly from hands-on access to this system and from close collaborations with the local computational science user community. We describe an example of this computer science research, in the area of dynamically resizable parallel applications.

1 Introduction

The importance of high-performance computing to computational science and engineering is widely recognized. Recent high-profile reports have called for greater investments in HPC and in training the next generation of computational scientists [1, 2]. Meanwhile, the raw power of the worlds fastest supercomputers continues to grow steadily, relying primarily on Moores Law and increasing processor counts. However, the appetite of computational science and engineering (CSE) researchers for high-end computing resources seems to grow faster than the ability of high performance computing (HPC) centers to meet that need. Every science and engineering discipline is making greater use of large-scale computational modeling and simulation in order to study complex systems and investigate deep scientific questions. The number of CSE practitioners is growing, and the computational demands of the simulations are growing as well, as more accurate simulations of more complex models are attempted.

Lack of access to the very high end of HPC resources is a challenge to the development of a broad CSE community. Elite researcher groups will always have access to the worlds most capable machines, and rightfully so; but identifying which groups or problems deserve that access is an inexact science. Advances often come from unexpected places. The CSE community would greatly benefit if HPC resources were available to a much broader audience, both in terms of

V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3515, pp. 60–67, 2005.

solving important problems today and in terms of training the next generation of CSE practitioners. Traditionally, the most powerful HPC resources have been located at government laboratories or federally funded supercomputer centers. Access to federal laboratory facilities generally requires collaboration with a laboratory scientist. Access to supercomputer centers is by grant processes; but these centers are often oversubscribed, and tend to favor capacity computing rather than capability computing. Meanwhile, because the most capable HPC resources are hidden behind a fence, and because the CSE practitioner community is limited in its access, computer science research in HPC systems has often been disconnected from real HPC facilities and practitioners. Although there are several notable exceptions (e.g., vectorizing compilers, computer architecture, grid computing), we believe that many areas of CS systems research have not had close day-to-day contact with the wide range of resources and practitioners in high-end CSE. None of this is surprising, of course. Historically, the most powerful HPC resources have been extremely expensive to build and operate. Hence, it makes sense that supercomputers are located where they are, and managed the way they are: they have been too precious to locate on a college campus or to replicate across many campuses; and they have been too precious to let CS systems researchers spend time on them exploring new programming models and tools, communication layers, memory systems, runtime libraries, etc.

In this paper we describe an ongoing project at Virginia Tech which seeks to address the problems described above. Our goal has been to demonstrate that world-class on-campus supercomputing is possible and affordable, and to explore the resulting benefits for our academic community—a community consisting of both computational scientists and computer science researchers and students. In Section 2 we describe the goals, design and implementation of System X. Section 3 describes one of the CS research projects that is motivated and enabled by System X. We conclude with some lessons learned in Section 4.

2 System X

Planning for Virginia Tech's Terascale Computing Facility (TCF) [3] began early in 2003. The goal was to support a rapidly growing CSE program by bringing high-end supercomputing to campus. The key challenge was affordability. While supercomputers have been invaluable to numerous science and engineering fields, their high cost—tens to hundreds of millions of dollars—has limited deployment to a few national facilities. We sought to develop novel computing architectures that reduce cost, time to build and maintenance complexity, so that institutions with relatively modest budgets could acquire their own very high-end resource.

When deployed in November 2003, the original System X consisted of 1100 Apple G5 nodes, each with two 2 GHz IBM PowerPC 970 microprocessors. That machine was ranked #3 in the 22nd Top 500 list. In October 2004 the system was upgraded, with 2.3 GHz dual Xserve G5s replacing the original nodes. This system ranks 7th on the 24th Top 500 list; it is the worlds most powerful academic machine. The system has 4.4 TB of main memory and 88 TB

of hard disk storage. The cluster nodes are interconnected with two networks: a primary InfiniBand network and a secondary Gigabit Ethernet fabric.

System X is enabling computational science and engineering researchers to tackle a wide variety of fundamental research problems, including molecular modeling, quantum chemistry, geophysics, fluid dynamics, computational biology and plasma physics. System X also provides a versatile environment for research in supercomputer systems design. Experimental cycles are set aside to enable researchers to study programming models, operating systems design, memory models, high performance networking, fault tolerance and design of distributed data storage systems.

2.1 Originality

System X was novel in several ways. First, although large-scale clusters are not new, achieving our price/performance design goals required an architecture based on untested cutting-edge technologies. None of System X's components—the Apple G5, the IBM PowerPC 970 processor, the Infiniband interconnect, the OS X operating system, and the Liebert hybrid liquid air cooling system—had ever been deployed at this scale. Furthermore, new systems software had to be written to enable the individual nodes to act in concert to form a tightly coupled supercomputer. The second novel aspect of System X is the speed at which it was constructed. Typical supercomputers of this class take eighteen months in the design and construction phases. Since our goal was to improve price/performance, we had to significantly reduce the design and build time in order to get the best performance for our limited budget. System X was designed, built and operational within three months. The third novel aspect is the cooling system. Typical supercomputing facilities and large data centers use air-conditioning technology to cool their facilities. Since the original System X consisted of 1100 nodes in a 3000 sq. ft. area, it generated a very high heat density making an air-conditioning based cooling technology very inefficient. Researchers from Liebert and Virginia Tech developed and deployed a liquid/air cooling system that uses chilled water and a refrigerant piped through overhead heat-exchangers. In this domain, the liquid air cooling technology is significantly cheaper and easier to deploy and maintain when compared to air-conditioned systems. This is the first deployment of this cooling technology.

Finally, as todays computational clusters evolve into tomorrows national cyber infrastructure, a key issue that needs to be addressed is the ability to mask component failures endemic to any large-scale computational resource. While previous generations of supercomputers engineered reliability into systems hardware, todays largest HPC platforms are based on clusters of commodity components, with no systemic solution for the reliability of the resource as a whole. For instance, if a supercomputer design is based on thousands of nodes, each of which fails only once a year, the system as a whole will fail multiple times per day. We have developed the first comprehensive solution to the problem of transparent parallel checkpointing and recovery, which enables large-scale supercomputers to mask hardware, operating system and software failures—a decades old problem.

Our system, Deja vu, supports transparent migration of subsets of a parallel application within cluster and Grid infrastructures. This enables fluid control of dynamic computational resources, where subsets of jobs transparently migrate under the control of resource aware scheduling mechanisms and distributed administrative control.

2.2 Successes and Challenges

System X achieved its design goals: price, performance, and design/construction time. The system began operations exactly three months from the Apple G5 product announcement and within three weeks of actual delivery of the nodes. The systems software stack, developed over a period of six weeks, involved researchers at Virginia Tech, Ohio State University, Israel and Japan. The completed software now provides an environment similar to other world-class supercomputers at a fraction of the cost ($5.2M), enabling researchers to port their CSE applications to System X.

The recent upgrade to the Xserve nodes has had several advantages. First, it reduces the size of the supercomputer by a factor of three, so that the machine now requires only about 1000 sq. ft. of area. Secondly, the new system consumes significantly less power than its predecessor. Third, it generates less heat, thereby reducing our cooling requirements. Fourth, the Xserve platform has automatic error correcting memory which can recover from transient bit errors. Finally, it has significant hardware monitoring capabilities—line voltages, fan speeds, communications—which allows real-time analysis of the health of the system.

Building any supercomputer presents severe logistical challenges in managing multiple aspects of the design and installation. First, funds have to be raised to finance the project. We had to articulate the research needs of the academic community and the benefits of on-campus supercomputing capabilities and make a case to administrators at Virginia Tech and the National Science Foundation to fund the project. Furthermore, with our limited track record in this area, we had to present a groundbreaking design that had the potential to succeed.

The construction of System X required detailed logistical planning and substantial infrastructure. Construction involved installing additional power and electrical equipment and cabling, bringing an additional 1.5MW of power into the facility, building and installing new cooling facilities (featuring two 125 ton water chillers), modifying the compute nodes to add communications equipment, installing communications switches and writing software to integrate the nodes into a supercomputer. In all, five equipment vendors, 80+ construction staff and 160+ student volunteers worked very hard to complete the project within three months.

Over 160 student volunteers helped in testing the nodes of the supercomputer and installing a communications card in each node. Five systems support staff helped in installing over 10 miles of high speed communications cables to interconnect the nodes. This work was completed within three weeks. Finally, we spent five weeks stabilizing and optimizing the supercomputer and writing systems software to integrate the nodes into a tightly coupled system.

3 Dynamically Resizable Parallel Applications

One of the computer science research topics we are investigating is that of dynamically resizable parallel applications. This work is directly motivated and enabled by access to an extremely large cluster such as System X. In this context, we are developing a programming model and API, data redistribution algorithms and a runtime library, and a scheduling framework. The motivation for this work stems from observations about usage of the terascale system. Given the scale of System X and the wide and unpredictable variety of jobs submitted, effective job scheduling is a challenging problem. Conventional schedulers are static, i.e., once a job is allocated a set of resources, it continues to use those resources until the end of execution. It is worth asking whether a dynamic resource manager, which has the ability to modify resources allocated to jobs at runtime, would allow more efficient resource management. In related contexts, dynamic resource management has resulted in better job and system performance (e.g., [4, 5]). Dynamic resource management enables more fine-grained control over resource usage. With dynamic resource management, resources allocated to a job can change due to internal changes in the job's resource requirements or external changes in the systems overall resource availability. In our context, dynamic resource management would extend flexibility by enabling applications to expand to a greater set of resources to take advantage of unused processors. Running applications could also shrink to a smaller subset of resources in order to accommodate higher priority jobs. The system could change the resources allocated to a job in order to meet a QoS deadline. Such a system, which enables resizing of applications, can benefit both the administrators and the users. By efficiently utilizing the resources, jobs could be completed at a faster rate, thus increasing system throughput. At the same time, by enabling applications to utilize resources beyond their initial allocation, individual job turnaround time could be improved. With this motivation in mind, the focus of our research is on dynamically reconfiguring parallel applications to use a different number of processes, i.e., on "dynamic resizing" of applications.

Additional infrastructure is required in order to enable resizing. Firstly, we need a programming model that supports resizing. This programming model needs to be simple enough so that existing code can be ported to the new system without an unreasonable re-coding burden. Secondly, runtime mechanisms to enable resizing are required. This includes support for releasing processors or acquiring new processors, and for redistributing the application's state to the new set of processors. Algorithms and a library for process and data re-mapping are described in detail in [6]. Thirdly, we require a scheduling framework that exploits resizability to increase system throughput and reduce job turn around time. The framework should support intelligent decisions in making processor allocation and reallocation in order to utilize the system effectively—by growing jobs to utilize idle processors, shrinking jobs to enable higher priority jobs to be scheduled, changing resource allocations to meet QoS deadlines, etc. In our approach the application and the scheduler work together to make resizing decisions. The application supplies preferences for the number of processors and for

processor topology; the scheduler records performance data for this application and other applications running on the system. We have extended an existing parallel scheduler [7] to interact with applications to gather performance data, use this data to make decisions about processor allocation, and adjust processor allocations to maximize system utilization. The new components in our scheduling framework include a Job Monitor, Remap Scheduler (RS), Performance Data Gatherer (PDG), and a Resize Library.

Or prototype implementation targets applications whose computation time are dominated by large ScaLAPACK [8] matrix operations. The BLACS communication layer of ScaLAPACK was modified to support dynamic process management (using MPI-2) and data and processor topology remapping. We assume that the computation is iterative, with one or more large numerical linear algebra computations dominating each iteration. Our API gives programmers a simple way to indicate "resize points" in the application, typically at the end of each iteration of an outer loop.

At resize points, the application contacts the scheduler and, if possible, provides performance data to the scheduler. Currently the metric used to measure performance is the time taken to compute each iteration. The PDG, which stores performance information for all applications currently running in the system, gathers the performance data provided by the application. This data is used to make resizing decisions. When the application contacts the scheduler, the RS makes the decision of whether to allow the application to grow to a greater number of processors, shrink the set of processors allocated to a job and reclaim the processors to schedule a different application, or permit the application to continue at its current processor allocation. A decision to shrink may be made if the application has grown to a size that has not provided a performance benefit, and hence the RS asks the application to shrink back to its previous size. An application can also be asked to shrink if there are applications waiting to be scheduled. The RS determines which running applications it needs to shrink so that an estimate of the penalty to system throughput is minimized. The RS can also allow the application to expand to a greater number of processors if there are idle processors in the system. If the RS cannot provide more processors for the application, and it determines that the application does not need to shrink, it allows the application to continue to run at its current processor allocation. The resize library, which is linked to the application, is used to perform data redistribution and construction of new processor topologies when the RS asks the application to shrink or expand. After the resize library has performed the resizing of the application, the application can perform its computation on the new set of processors. This process continues for the lifetime of the application.

We are exploring various heuristics for deciding when to resize an application, and by how much. One simple idea is to use dynamic resizing to determine a "sweet spot" for a given application. The optimal number of processors on which to run a given application is almost never known a priori. In fact, the definition of "optimal" depends on whether one cares more about throughput for a mix of jobs or turn-around time for a single job. We have implemented a job resizing

algorithm that gradually donates idle processors to a new job and measures the relative improvement in performance (measured by iteration time), in an effort to estimate the marginal benefit (or penalty) of adding (or subtracting) processors for a given application. Given this information about several long-running jobs in the system, the scheduler can than expand or contract jobs as needed, either to efficiently soak up newly available processors, or to free up under-used processors for new jobs. Simple experiments show at least a 20% improvement in overall throughput, as well as improved turn-around time for at least half of the individual jobs (none took longer) [9]. These results are highly dependent on application characteristics and job mixes of course. But the intuition behind the improvements is clear: individual jobs benefit if they would otherwise have been running on too few processors, and the entire job set benefits because the machine is utilized more effectively.

4 Conclusions

Although System X has only been in full production mode since January of 2005, we are already seeing evidence of the benefits for computer science and computational science research and graduate education. An interdisciplinary community of researchers is emerging, including both core CS investigators and applications specialists from a wide range of disciplines, including computational biology, chemistry, mechanics, materials science, applied mathematics and others. The four most important benefits are listed below, none of which would likely occur without the on-campus availability of System X:

1. **Closing the loop.** Bringing CSE practitioners and CS systems researchers together around a single high-profile resource leads to useful synergies among the two groups. CSE researchers communicate their needs to computer scientists and benefit from new systems and algorithms; CS investigators use real CSE problems and codes as motivation and test-cases for new research.
2. **Opportunities for CS research.** Research in systems and algorithms for highly scalable computing requires flexible access to a machine of such scale. Without the affordability and accessibility of Systems X, work such as that described in Section 3 would not be happening.
3. **Capability computing.** We have maintained our emphasis on capability computing, as opposed to pure capacity computing. We reserve time on the system for applications that require a substantial part of the machine for a substantial amount of time—computations that simply could not be done without this resource. The affordability and the relatively small user community, compared to national computing centers, makes this possible.
4. **Opportunities for students.** Dozens of graduate and undergraduate students are gaining valuable experience with HPC and CSE research and operations. The system provides a unique training opportunity for these students.

Despite these encouraging signs, there are still challenges facing us as we pursue the original goals of System X. In the first place, preserving the dual

mission of the facility (both computer science research and CSE applications) is not always easy, both politically and technically. Politically, we have to continue to demonstrate that experimental computer science research is as important as traditional CSE applications, both as quality research and for its benefits to future HPC applications. Technically, the challenge is to develop job management strategies and infrastructure to support both modes. For example, experimental computer science research projects may need many processors for a short period of time, while CSE applications may need a modest number of processors for a very long time. We are leveraging our fault-tolerance work to enable a sophisticated suspend/resume mechanism which allows very long-running applications to be suspended briefly, to allow experimental jobs to access the machine.

A second challenge is in finding the appropriate level of support for users. We have intentionally kept the TCF's staffing level low, in part to keep cost-recovery needs low. This means that relatively experienced HPC users get the support they need, but we do not provide extensive application-level help or help in parallelizing existing sequential codes. We do provide basic training in MPI and in using the system. Since parallelization is usually best done by someone who knows the code already, we are working to equip Virginia Tech CSE research groups to parallelize their own codes, if they so desire. This is not always an easy argument to make, however. For example, some science and engineering disciplines are slow to give credit to graduate students for writing parallel codes.

References

1. Report of the High End Computing Revitalization Task Force (HECRTF), http://www.itrd.gov/hecrtf-outreach/
2. Science and Engineering Infrastructure for the 21st Century, National Science Board, http://www.nsf.gov/nsb/documents/2003/start.htm
3. Terascale Computing Facility, Virginia Tech, http://www.tcf.vt.edu/
4. Moreira, J.E., Naik, V.K.: Dynamic Resource Management on Distributed Systems Using Reconfigurable Applications. IBM Research Report RC 20890, IBM Journal of Research and Development **41** (1997) 303–330
5. McCann, C., Vaswami, R., and Zahorjan, J.: A Dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors. ACM Trans. Comput. Syst. **11** (1993) 146–178
6. Chinnusamy, M.: Data and Processor Re-mapping Strategies for Dynamically Resizable Parallel Applications. MS Thesis, Virginia Tech, Dept. of Comp. Sci., 2004.
7. Tadepalli, S., Ribbens, C. J., Varadarajan, S.: GEMS: A job management system for fault tolerant grid computing. In: Meyer, J. (ed.): Proc. HPC Symposium. Soc. For Mod. and Simula. Internat., San Diego, CA (2004) 59–66
8. ScaLAPACK Project, http://www.netlib.org/scalapack/
9. Swaminathan, G.: A Scheduling Framework for Dynamically Resizable Parallel Applications. MS Thesis, Virginia Tech, Dept. of Comp. Sci., 2004.