Building a large low-cost computer cluster with unmodified Xboxes

B.J. Guillot, B. Chapman and J.-F. Pâris
Department of Computer Science
University of Houston
Houston, TX, 77204-3010
bguillot@acm.org, {chapman, paris}@cs.uh.edu

Abstract

We propose to build a large low-cost computer cluster in order to study error recovery techniques for today and tomorrow's large computer clusters. The Xbox game console is an inexpensive computer whose internal architecture is very close to that of a conventional Intel-based personal computer. In addition, it can be rebooted as a Linux computing node through software exploits without having to purchase any additional hardware or even opening the Xbox. We built a four-node cluster consisting of four unmodified Xboxes running Debian Linux and found out that a cluster of Xboxes linked by a Fast Ethernet would constitute a scaled down version of a current generation supercomputer with the same number of nodes. As a result, it would provide a cost-effective testbed for investigating novel distributed error-recovery algorithms and testing how they would scale up.

Keywords: computer clusters, game console, Linux, distributed error recovery.

1. Introduction

Improvements in technology as well as pricing trends have enabled the construction of clusters with increasingly larger node counts. However, the reliability of the cluster decreases in rough correspondence to the number of configured nodes, greatly reducing the attractiveness of such systems, especially for large or long-running jobs.

In general, dealing with faults is a matter for the application developer, who must insert checkpoints into each application and thus periodically save all pertinent data. This is necessarily costly, both in terms of time required and in the amount of storage space used. As a result, a number of researchers have begun to consider alternative approaches to checkpointing, including system-initiated checkpointing, libraries incorporating checkpointing and automatic support for this via the compiler.

All these approaches raise scalability issues that should be resolved by testing them on large to very large clusters. Unfortunately, this is rarely possible because very large clusters are too expensive for most research groups. Hence most of the current research has to be done on relatively small clusters comprising only a limited number of nodes.

We propose to lift this limitation by building a testbed cluster consisting of large numbers of very cheap nodes. Since this cluster would be expressly designed for experimentation, its cost per node would be of more import than its cost per MFLOP. We further propose to build this testbed cluster using unmodified Xbox game consoles. In addition to its low price, this console offers the double advantage of not requiring any additional hardware to operate as a computing node and having an internal architecture very similar to that of a conventional personal computer.

The remainder of this paper is organized as follows. Section 2 reviews the Xbox architecture and explains how to convert it into a Linux computer without any hardware modification. Section 3 describes our experimental Xbox cluster. Section 4 discusses the intended uses of our cluster. Section 5 mentions relevant work. Finally, section 6 has our conclusions.

2. The Xbox

The Microsoft Xbox is a video game console that currently sells for $180 in consumer electronic stores. It has an Intel Celeron processor, a hard disk, and Fast Ethernet. Embedded security features prohibit execution of software other than authentic games, but persistent users have found ways to boot Linux using either software-only exploits or hardware modification (ROM replacement).

The CPU is a custom version of Intel's 733 MHz Celeron microprocessor. While conventional Celeron microprocessors of this speed would have a 66 MHz Front Side Bus (FSB) and a 64 KB L2 cache, the Xbox Celeron has a 133 MHz FSB and a 128 KB L2 cache. Due to the faster FSB, processor performance closely matches that of an equivalent speed Pentium III.

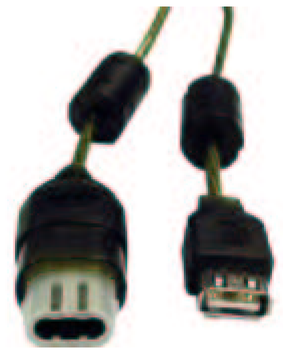

Figure 1. Xbox USB converter cable [1].

At first glance, the Xbox may not appear to be a computer because it has no serial, parallel, or PS/2-compatible keyboard and mouse ports. The omissions of these ports and the Floppy Drive Controller (FDC) actually help solidify the position that the Xbox is a modern “Legacy-Free” PC [2, 3]. Four USB 1.1 ports on its front panel are conveniently disguised: they require a special connector to hook a peripheral to it (see Figure 1).

NVIDIA designed the NV2A Northbridge memory controller and MCPX Southbridge (Media and Communications Processor for Xbox). This chipset is very similar to NVIDIA’s nForce chipset found in the retail channel, except it has a hidden 512-byte startup code that has a sole purpose of checking the currently installed ROM to verify it has a proper Microsoft cryptographic signature.

The console has 64 MB of surface-mount DDR SDRAM. The NVIDIA GeForce3 Graphics Processing Unit (GPU) does not have a separate pool of video RAM, and must share this memory. The physical memory size can be upgraded to 128 MB for those with exceptional soldering abilities [4].

Each unit has at least an 8 GB IDE hard disk, further divided into several small partitions for system information (500 MB) and caches (three 750 MB partitions). Ample space is available for user data in a 5 GB partition reserved for saved game sessions. The hard disk is locked with a password unique to an individual Xbox. It cannot be read using a regular PC unless the disk is first unlocked, and then, only when used with software that supports the non-standard FATX file system.

The Xbox operating system fits in a 256 KB ROM, and is based on an embedded version of the Windows 2000 kernel. Strong cryptography is used to allow only Microsoft-signed applications to run.

We note that the Xbox uses a 2048-bit RSA key length while most financial institutions use 1024-bit keys. Because all applications must be signed, this means it is not possible to execute homegrown code on an Xbox without using some kind of software exploit or hardware modchip.

Software Exploits

On March 29, 2003 an anonymous hacker known only as Habibi-Xbox, posted a message in an online forum claiming he had figured out how to start Linux on an unmodified Xbox. A file attached to his message demonstrated the technique. His results were confirmed, and it was reported that Habibi might have won the $100,000 challenge sponsored by Lindows founder Michael Robertson [5]. The code was so well obfuscated [6], that it took several months before his actual exploit code was found encrypted in the JPEG header.

This exploit is in the form of a "game-save" for Electronic Art's *007: Agent Under Fire*. Habibi's file is first placed on a memory card and transferred to the Xbox saved game partition. The 007 game is placed in the DVD tray, which restarts the console. The game menu appears shortly thereafter, enabling the user to select the option to load a previously saved game. This action initiates the buffer overflow attack, and the exploit takes control. The security features of the Xbox do not realize a different application is running (the Linux bootloader), and it allows the code to continue to execute, all the time believing it is still running authorized code with the cryptographic signature of the 007 game.

The major drawback of this exploit is that Linux can only start after the user inserts the 007 game and uses the game pad controller to manually initiate the load of the game-save. A television must be attached to the unit to see what you are doing. The ideal scenario is to boot directly into Linux bypassing the need for the 007 game, game pad, and television set.

When the Xbox is powered on without a game in the DVD tray, a program known as the Dashboard starts. The Dashboard is a Microsoft program that lets you configure the console's date and time as well as transfer files from memory cards to the game-save partition. The Dashboard loads a set of font files when it initializes in order to render on-screen messages to the user.

Figure 2. Sony Linux Kit for PlayStation 2 [8, 9]

A second exploit, the “font exploit,” involves overwriting the existing font files with special versions that contain code allowing the Xbox to run Linux upon power-on. The font files are located on the system partition, so the Xbox must already be running Linux (by using the first exploit) before the files can be transferred using FTP.

One of the design flaws in the console is that cryptographic signatures are only required on application executables. Signatures are not required on fonts and audio files. Because of this, any executable code placed inside of a font file can get executed provided the appropriate preamble is present that induces a buffer overflow.

By using the game-save exploit to install the font exploit, the Xbox has now become a stand-alone, bootable Linux computer. The box never needs to be opened. No soldering is done. No modchips ever had to be installed.

Table 1. Xbox Linux versus PlayStation 2 Linux.

System	Microsoft Xbox	Sony PlayStation 2
CPU	733 MHz Intel Celeron/P3 133 MHz FSB 16 KB L1 cache 128 KB L2 cache MMX and SSE instructions	"Emotion Engine" 300 MHz MIPS R3000 compatible Full MIPS-3 instruction set with extensions from MIPS-4 and –5 Two Vector Processing Units (VPU)
Memory	64 MB DDR SDRAM	32 MB RDRAM
Network	100 Megabit/sec	Sold* with Sony Linux Kit (100 Megabit/s)
Hard Disk	8 or 10 GB	Sold* with Sony Linux Kit (40 GB)
Removable media	DVD-ROM/CD-ROM	DVD-ROM/CD-ROM
Special features	Programmable color LED Temperature sensors	N/A
Ports	Four USB ports *(non-standard connectors)*	Two USB ports One Firewire (not accessible from Linux)
Console Cost	$180	$180
Extra Costs*	N/A	$200 Sony Linux Kit (HD, Ethernet) Sony 8 MB memory card (price varies)
Total Cost	$180	At least $380

Comparison with the Sony PlayStation 2

The Xbox comes with all the hardware needed to run Linux out of the box. The PS2 requires that the user buy a special $200 Linux kit (Figure 2) that includes the required software, a hard disk, and an Ethernet adapter. See Table 1 for a hardware comparison of the two platforms.

The Xbox has a faster processor and twice as much RAM as the PS2. Because of this, the claim can be made that the Xbox is the better choice for a scientific computing console. Since the Xbox uses a fairly standard 100% Intel compatible processor, actual Linux binaries can run. All the software that runs

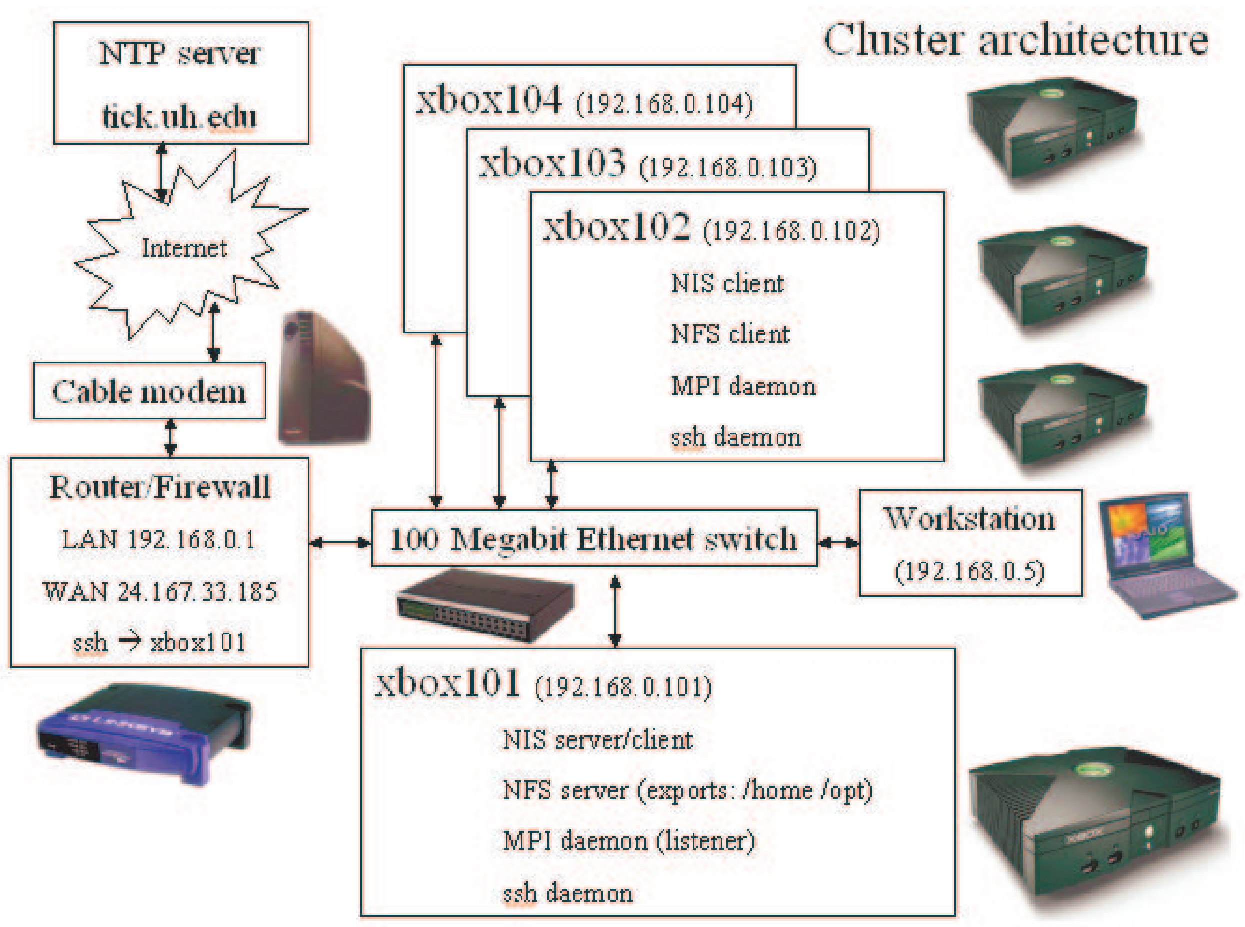

Figure 3. Four-node Xbox cluster network diagram.

on a Linux workstation PC will run on a Linux Xbox (assuming it has enough RAM). Since the PS2 is using a very custom MIPS chip, binaries for other MIPS platforms cannot run without being re-compiled from the source code. On the other hand, the PlayStation includes two Vector Processing Units (VPU), for which extremely high performance code can be written.

3. The Xbox Cluster

We built our Xbox cluster by successively installing Linux on our four Xboxes. We started by configuring the first node as the master node and configured then a slave node that we cloned on the third and fourth cluster nodes. The cluster network diagram can be seen in Figure 3.

We used a special distribution of Linux based on the well-known Debian distribution. This special distribution was tailored especially for device drivers needed by the Xbox. Its standard device drivers are NVIDIA chipset Linux drivers. It also includes special drivers that support the temperature sensors as well as drivers allowing the X window system to work over a television output.

We further customized that distribution by removing all unneeded daemons. FTP, Telnet, printer, and related daemons were turned off and the X window automatic startup was disabled. This increased both the cluster security and the amount of available memory at startup. As a result, the only way to login to the system is using the ssh terminal emulator. Using ssh rather than telnet and ftp was not an easy decision. Security is clearly better in ssh, but it has a much greater memory footprint than telnet and ftp. Security is clearly non-existent in telnet and ftp since passwords are sent over sockets in plain text. We decided that security was more important than memory footprint and selected ssh.

The master node of our cluster runs a NFS (Network File System) server, NIS server (and client), MPI daemon (in listener mode), and an ssh daemon. The /home and /opt directories are exported via NFS. The /home tree contains all of the users home directories and is a separate 2 GB partition on the master Xbox. The /opt tree contains all of the optionally installed software such as the Intel C/C++ and Fortran compilers, MPICH, Java, BLAS libraries, etc.

The NFS clients on all the slave nodes import the /home and /opt trees so that a user can ssh into any node and see the same structure as the master node. This also allows MPI programs to run normally since they need to have the same file system visibility. The slave nodes are NIS (Network Information Service) clients, which allow user accounts, groups, and passwords to be shared between all nodes in the cluster. The slave nodes also run a MPI daemon (but not in listener mode).

A cron job runs every 15 minutes and examines the temperatures on each node. The power supply fan speed will then adjust to keep the temperatures reasonable.

The only node that allows inbound Internet connections (ssh) is the master node. Outbound Internet traffic is allowed on all nodes. In fact, all nodes routinely pool a local NTP (Network Time Protocol) server to synchronize the system clock to the current time. Hence all slave nodes must be always accessed by doing first a secure shell login on the master node and then doing another secure shell login from the master.

Cluster Performance

We measured the performance of our cluster using two well-known benchmarks, namely, Linpack (a linear algebra package that is the industry standard benchmark for parallel computers) and mprime (a program that takes very large numbers and tries to determine if the number is prime or composite).

Linpack

We selected Linpack because it is used to determine how supercomputers stack up against each other in the Top 500 list. Linpack is MPI-enabled, and can be run on a single node, or its workload can be spread across all nodes in the cluster. The actual linear algebra package can be changed at will and dynamically linked to at run-time. We linked it with the self-optimizing BLAS (Basic Linear Algebra) package. BLAS was compiled from source code on the Xbox. Note that the BLAS build procedure is self-optimizing as it builds several versions that are optimized for different cache sizes, run the results, and then pick the best of the lot and use that version as the final pick.

The best Linpack results for a single node operating alone was 0.5 Gigaflops/second. The best result for all four nodes operating as a single parallel computer was 1.4 Gigaflops/second.

mprime

Unlike Linpack, the mprime benchmark runs only on a single node. Its algorithm changes depending upon what kind of processor it is running on. It will use double precision arithmetic on Pentium 3 compatible processors (like the Xbox's Celeron, and the AMD Athlon), and vectorized SSE2 double precision arithmetic on Pentium 4 processors. As a result, mprime results will favor the Pentium 4 by a large margin. The core of the mprime algorithm involves squaring very large numbers using a Fast Fourier Transform operation. Figure 4 shows results comparing the Xbox with two desktop computers. As we can see, the Xbox CPU appears to be about half as fast as an Athlon XP 2000+ CPU. These numbers appear reasonable as the Athlon XP 2000+ runs at 1.67 GHz, which is more than twice the clock

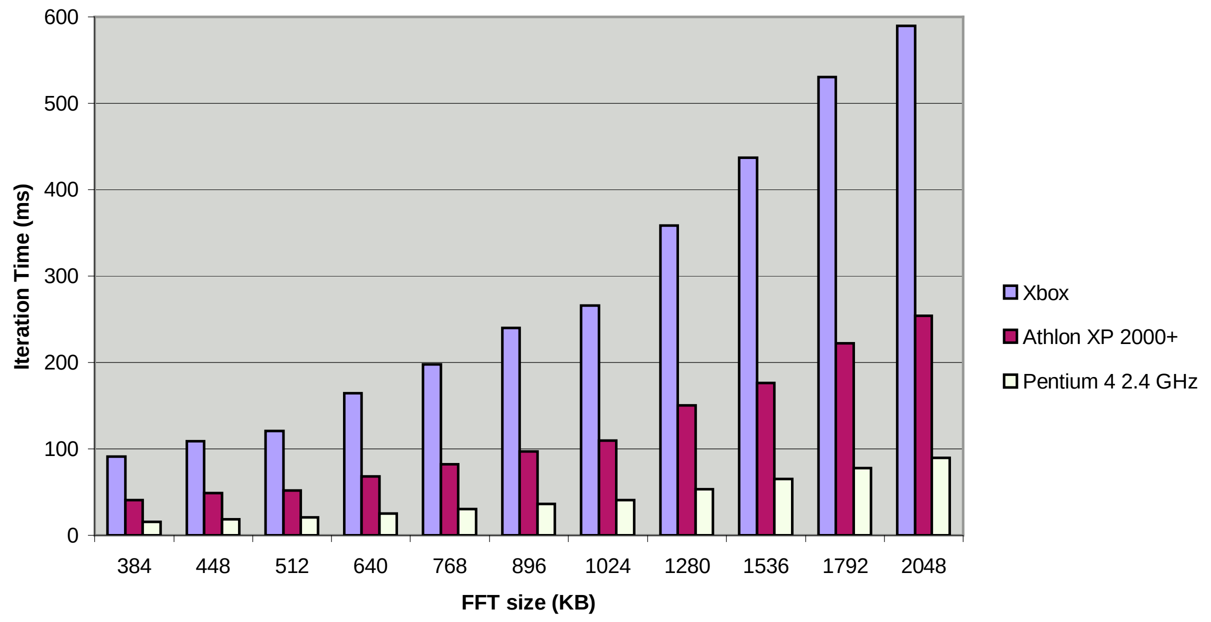

Figure 4. Benchmark results for mprime (lower times are better)

rate of the Xbox CPU. The Pentium 4 numbers are much higher due to the use of the SSE2 instruction set, and the Xbox does not even come close to matching them.

Table 2. Power Usage at 100% CPU load

(Normalized to Xbox, smaller numbers are better)

700 MHz Pentium III Laptop	0.6
733 MHz Xbox	**1.0**
1 GHz Athlon Desktop	1.8
1.67 GHz Athlon Desktop XP 2000+	2.0

Power consumption

One of the big issues when building a cluster is power consumption. Most homes and businesses have 15-20 Amp circuits [7] that can drive 1500-2000 Watts at 120 Volts. The less power each node requires, the more nodes can be safely put on a single circuit. Table 2 shows the power usage of various workstations relative to the Xbox. We obtained these numbers by measuring the "percent loads" on a CyberPower 1500AVR uninterruptible power supply (UPS) when each device was plugged into it and converting later these percentages to relative numbers for comparison purposes.

It is interesting to note that the Xbox consumed less power than the Athlon desktops it was compared against, but consumed more power than a similarly powerful Pentium III laptop.

Discussion

Our performance benchmarks demonstrated that the Xbox performed competitively against Pentium III CPU's but not against Pentium 4. That microprocessor has an SSE2 instruction set allowing extremely efficient double-precision number crunching, at least when it is used with compilers that optimize for the chip architecture.

In a real parallel cluster using thin 1U (or smaller) blades plugged into a rack with high-speed Pentium 4, Xeon, or Athlon processors, each node may cost anywhere from \$1000-\$2000. A 128-node system would easily cost \$150,000. If we assume that the nodes of this hypothetical cluster have CPUs ten times more powerful than the Xbox Celeron, Gigabit Ethernet connections ten faster than the Xbox Fast Ethernet, and memory sizes ten or twenty times larger than the Xbox's 64 MB, we could say that the Xbox is a roughly 1:10 scale version of a current generation supercomputer node.

We can thus envision an Xbox cluster as a scaled-down version of a production supercomputer. It will not be as fast, but it could easily have the same amount of nodes in order to test and tune the performance of parallel algorithms before deploying them to the production cluster.

This approach is not perfect. First, the disk access times of both clusters will be roughly equivalent. Second, it would not apply if the parallel algorithm relies on both shared memory and distributed memory approaches. Many of the modern cluster nodes consist of at least two processors. The Xbox would not be able to simulate this scenario very well if the algorithm was dependent upon the symmetric multiprocessing (SMP) capabilities of the node. A further complication is that modern nodes sometimes have more than one Ethernet port. Two or more ports of course mean two or more paths of data that flow out from the node. This would be difficult if not impossible to model.

Remaining Issues

There are three significant remaining issues with our Xbox cluster. These are reboot difficulties, A/V cables, and continuous hardware changes that appear in new models of the console.

Rebooting difficulties

The first issue involves recurring problems that occur while rebooting the box. In the Xbox community, this issue is referred to as the “clock loop” problem. On some occasions, when the console is rebooted, the machine can get “stuck” in a seemingly endless reboot loop. This is only a minor problem, because the machine will always come out of the loop (sometimes in less than a minute, in other times, it may take over an hour). Because you expect high availability with all cluster nodes, it is unlikely that reboots will be required once the cluster is operating in a steady-state scenario. This will only be an issue if steady state operation requires scheduled reboots, although a properly administered cluster should not need reboots. The cluster nodes appear very stable, and we have observed individual node uptimes surpassing seventy days.

Audio/Video cables

The second issue is more a minor annoyance than a real problem. The Xbox can easily run Linux without a monitor as long as it has that audio/video cable connected to the back of the console. That cable needs not to be connected to any other device and can indeed hang off the back. This is annoying because the long A/V cables take up considerable space and can cause heat flow issues in a stack of consoles if care is not taken to place the A/V cables in an optimal spot. The BIOS prevents the Xbox from booting without an A/V cable attached. Since the BIOS cannot be modified without opening the box and soldering, we have to live with the dangling A/V cables if we are to stay true to our “unmodified Xbox” philosophy: individuals using modchips have been able to install new versions of the BIOS that allow booting without needing A/V cables plugged in.

Minor hardware changes

Our last issue involves minor hardware changes that have occurred over the lifetime of the Xbox. Microsoft has manufacturing plants in three countries [10]: China, Mexico, and Hungary. The assembly lines switch components [11] more frequently than we expected. At least three different DVD drives have been used (Thompson, Philips, Samsung). Two different kinds of hard disks have been used (Western Digital 8 GB, Seagate 10 GB). The video encoder chip has switched [12] from Conexant to

Focus in recent versions. Any hardware change could conceivable require a change to the Xbox Linux distribution to bundle updated drivers for the new hardware. Eventually the community will catch up, but the risk is that if you were to buy dozens or hundreds of new consoles, they may not yet work with the current Linux distributions or exploits.

4. Future Work

We intend to use this system to study fault tolerance issues, with a focus on adaptive automated handling of fault tolerance. In such an approach, system parameters are evaluated to determine the observed failure rate and to draw conclusions about the frequency of data collection needed to enable restart. We use our enhanced version of the Open64 compiler infrastructure to experiment with automation of this process with MPI codes.

Based on the technique in which the state of a process is saved and resumed on restart, checkpointing and migration solutions can be classified into system-level and application-level solutions. In system-level solutions, the whole process image of the checkpointed application is saved on non-volatile media and will be reloaded on a restart or migrated-restart. Such checkpointing is normally transparent to the application. However, such transparency puts strict restrictions on the system calls that may be used in the application, or requires that the operating system kernel be patched. A comprehensive survey of these kinds of solutions can be found in reference [13].

For message-passing parallel applications, one must decide whether checkpointing is coordinated or uncoordinated, and whether it is blocking or non-blocking. Handling dangling messages during checkpointing is a major challenge. For example, Condor's checkpointing solution [14, 15] only works for certain kind of processes that do not call fork(), exec() or have any communication via pipes or files. Such restrictions are impractical for many parallel applications. Co-Check MPI [16] was the first MPI implementation that used the Condor library for checkpointing an entire MPI application. Co-Check processes flush their message queues to avoid in-flight messages getting lost, and then synchronously checkpoint, which results in large datasets for a single checkpoint.

Application-level checkpointing [17] provides the benefit of saving only the minimal state needed to recover instead of the entire process image. This approach inserts checkpoint calls in application source code that will save only the required application data on a checkpointing request. This can only be done via user-inserted calls, unlike the former approaches where an application can be checkpointed at almost any point of execution. The programmer controls where to insert checkpoints and which data need to be saved, which is a disadvantage as it complicates the coding of the program. DataGrid [18] uses this technique to checkpoint and save the application relevant data in a {variable, value} pair format. In compiler-automated application-level checkpointing [19], checkpointing calls are inserted by a pre-processor, which is transparent to the application programmer. In order to minimize the cost of coordinated checkpointing, the pre-processor tries to insert checkpoints around MPI barriers. This solution relies on the compiler to detect and insert suitable checkpoints, so the checkpoint library needs to be complete in order to handle saving of state at different checkpoint locations. Also it cannot guarantee an optimized state saving strategy. Another issue it has to take care of is how to save hidden MPI states.

A recent trend in application-level checkpointing is the idea of a reconfigurable or malleable application. A reconfigurable application can be stopped and restarted on a modified set of resources to accommodate dynamically changing availability of processors. Reconfiguration provides flexibility of scheduling and resource management as well as fault-tolerance since the application can be restarted in the event of processor or network failure. Also, if portability is supported, the checkpointed application or parts of it can be migrated to other available resources. Furthermore, if supported by a powerful scheduler and run-time systems with scheduling policy information and dynamic resource/application monitoring, such an application can be executed in a highly adaptive way. Some systems utilizing this idea are DyRecT [20], IBM DRMS [21], and SRS [22].

5. Related work

The largest game console cluster known to date is the more than seventy-node PlayStation 2 system [23] built by the National Center for Supercomputing Applications (NCSA) and the Computer Science department of the University of Illinois. The computer science department has succeeded in writing a very impressive, but highly customized, 1 GF/s matrix multiply routine [24] on the PS2. Their chemistry department [25] has experimented with porting computational chemistry codes to the PS2. Chemistry professor Todd Martinez even coined the term COTTS—for Commodity-Off-The-Toy-Shelf hardware—to describe the idea of building a computer cluster using video game consoles.

The largest Xbox cluster other than the one we implemented is the three-node system [26] built by Adam Cecchetti. His attempt used hardware modchips in order to run Linux. Installing a modchip requires opening the Xbox (which instantly voids the warranty), and attaching to the motherboard wires from a small daughtercard. This is a risky operation that can easily damage the Xbox. We never had to do that or even to open the Xbox as our implementation only relies on software exploits.

6. Summary and Conclusions

The Xbox game console is an inexpensive computer that can be used as a node in a cluster with relative ease. Since its internal architecture is very close to that of a conventional Intel-based personal computer, algorithms do not require any special optimization to run on it. A cluster of Xboxes linked by a Fast Ethernet would constitute indeed a scaled-down version of a current generation supercomputer with the same number of nodes while costing between one tenth and one fifth of its price. As a result, it would provide a cost-effective testbed for investigating novel distributed error-recovery algorithms and testing how they would scale up.

References

[1] http://image.lik-sang.com/images/170/xbox-usb_cable.jpg

[2] Legacy-Free Hardware and BIOS Requirements. http://www.microsoft.com/whdc/hwdev/archive/platform/pcdesign/LR/Lf.msp

[3] A. Green, M. Steil. The Xbox is a PC. February 21, 2003. http://xbox-linux.sourceforge.net/docs/xboxpc.html

[4] M. Steil. Running Linux on the Xbox. *Linux Journal*, Volume 2003, Issue 111. July 2003.

[5] D. Becker. Hacker Cracks Xbox Challenge. March 31, 2003. http://zdnet.com.com/2100-1103-994794.html

[6] Technical Analysis of 007: Agent Under Fire save game hack. July 9, 2003. http://xbox-linux.sourceforge.net/docs/007analysis.html

[7] R. Brown. Power and cooling for your Beowulf. May 12, 2003. http://www.phy.duke.edu/brahma/beowulf_book/node59.html

[8] Linux (for PlayStation 2) Version 1.0 FAQ. June 3, 2003. http://playstation2-linux.com/faq.php#What_is_Linux_for_PS2

[9] http://playstation2-linux.com/Linux_kit.jpg

[10] M. Steil. The Xbox manufacturing process. October 14, 2003. http://xbox-linux.sourceforge.net/docs/manufacturing.html

[11] M. Steil. Xbox hardware database. November 15, 2003. http://xbox-linux.sourceforge.net/docs/versionsdb.html

[12] Xbox Linux Project News Archive. http://xbox-linux.sourceforge.net/news_archive.php

[11] N. Stone, J. Kochmar, R. Reddy, J. Ray Scott, J. Sommerfield, C. Vizino. A checkpoint and recovery system for the Pittsburgh Supercomputing Center Terascale computing system. http://www.psc.edu/publications/tech_reports/chkpt_rcvry/index.html

[14] J. B. M. Litzkow, T. Tannenbaum and M. Livny. Checkpoint and migration of UNIX processes in the Condor distributed processing system. Technical Report 1346, University of Wisconsin-Madison, 1997.

[15] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, Condor-G: A computation management agent for multi-institutional grids. *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing* (HPDC10), San Francisco, California, August 7–9, 2001.

[16] G. Stellner. CoCheck: Checkpointing and process migration for MPI. *Proceedings of the 10th International Parallel Processing Symposium* (IPPS '96) April 15–19, 1996 Honolulu, HI <http://www.computer.org/proceedings/ipps/7255/7255toc.htm>

[17] A. Beguelin, E. Seligman, and P. Stephan. Application level fault tolerance in heterogeneous networks of workstations. *Journal of Parallel and Distributed Computing*, 43(2):147–155, 1997.

[18] Job Partitioning and Checkpointing: DataGrid. <http://www.pd.infn.it/~gianelle/datagrid/documents/DataGrid-01-TED-0119-0_3.pdf>

[19] G. Bronevetsky, D. Marques, K. Pingali, P. Stodghill. Checkpointing and communication: Automated application-level checkpointing of MPI programs. *Proceedings of the 9th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 84–94, June 2003, San Diego, California, USA

[20] Etienne Godard, Sanjeev Setia and Elizabeth White. DyRecT: Software Support for Adaptive Parallelism on NOWs. *Proceedings of the IPDPS Workshop on Runtime Systems for Parallel Programming*, Cancun, Mexico, May 2000.

[21] V. K. Naik, S. P. Midkiff, and J. E. Moreira. A checkpointing strategy for scalable recovery on distributed parallel systems. *Proceedings of SuperComputing (SC) '97*, San Jose, November 1997

[22] Sathish S. Vadhiyar and Jack J. Dongarra, SRS: A framework for developing malleable and migratable parallel applications for distributed systems. *International Journal of High Performance Applications and Supercomputing*, 2003 http://hipersoft.cs.rice.edu/grads/publications/vadhiyar-frameworkmalleable.pdf

[23] C. Steffen. Scientific Computing on the Sony PlayStation 2. November 17, 2003. http://arrakis.ncsa.uiuc.edu/ps2/

[24] C. Steffen. Scientific Computing on the Sony PlayStation 2: Using the Vector Units http://arrakis.ncsa.uiuc.edu/ps2/using_vector_units.php#1GF

[25] T. Martinez. Computational Chemistry on the Sony PlayStation 2. http://spawn.scs.uiuc.edu/research/sonyps2/ps2project.htm

[26] A. Cecchetti. XBOX Linux Cluster. http://www.shadowflux.com/xbox.html