Design and Evaluation of a 2048 Core Cluster System

Frank Mietke[1] Torsten Mehlan[1] Torsten Hoefler[1,2]
Wolfgang Rehm[1]

[1]Technical University of Chemnitz, Chemnitz, 09107 GERMANY
{mief,tome,htor,rehm}@cs.tu-chemnitz.de

[2]Open Systems Laboratory, Indiana University, Bloomington IN 47405, USA
htor@cs.indiana.edu

Abstract

Designing a 2048 core high performance cluster, including an appropriate parallel storage complex and a high speed network, under the pressure of limited budget (2.6 Mio Euro), performance, thermal and space limitations is really a challenging task.
In this paper, we present our design decisions and their reasons, our experiences during the installation stage as well as performance numbers using well-known benchmarks in the field of scientific computing, networking and I/O, and real world applications.

1 Introduction

Up to the beginning of the year 2000 the supercomputers in the TOP500 list[1], was dominated by Massively Parallel Processor (MPP) (51.6%) and Symmetric Multiprocessor (SMP) (33.8%) systems. Cluster systems (5.6%) played only a minor role. Since then the cluster architecture has become the dominant supercomputing platform (81.2% in 11/2007 issue of TOP500 list). This was mainly due to a growing PC/server market, which made the single machine more affordable, the broader support of Linux, the invention of high-speed networks as well as a growing software stack which simplified the setup, administration and programming [17, 27].

1.1 Supercomputing in Chemnitz

The growing complexity of scientific problems and the demand of more compute power led to the procurement of the first supercomputer at the Technical University of Chemnitz (TUC), a Parsytec GC 128 PowerPlus which was, in 1994, one of the fastest machines in Germany. After 4 years it had become obsolete and the need for a new system which had to be able to satisfy the growing needs of the steadily growing user community became evident. To achieve the best price performance ratio, the university computing center decided to design and build their own cluster computer from desktop PCs running Linux. Two independent Fast Ethernet networks served as high performance communication and administration mediums. This new system was named CLiC (Chemnitzer Linux Cluster) and was operational in 2000. This cluster was, in the Top500 metric, the fastest in Germany and the second fastest in the category self-made in the world after the CPlant/Siberia at the Sandia National Laboratories. Also the price-performance ratio was one of the best in the world.

The user community as well as the complexity of scientific problems grew further which led, 4 years later, to new discussions about an update of the supercomputer. A short overview of current projects is given in [1].
We will describe the design process, the hardware and software experiences with this new system, called CHiC (Chemnitzer High-Performance Cluster) in the following sections. Several results of synthetic benchmarks and real world applications are presented in Section 3 to assess the performance of the newly deployed cluster. We conclude the relevant results in Section 4 and outline the future work in Section 5.

2 CHiC

To further strengthen the HPC capabilities at the TUC and thus accelerate the scientific outcome, the CHiC was optimized for high-performance parallel computing as well

[1]bi-annual ranking of fastest supercomputers in the world, issue 11/1999

as high job throughput. In this section we are discussing the design, hardware, software and the first experiences with the new system.

2.1 Design

In the year 2000 the CLiC was an effort to build a big cluster (528 nodes) using desktop computers and a Fast Ethernet communication network. All nodes were connected through a single Fast Ethernet switch. To utilize the budget of 1.25 Mio Euro this cluster was self-made and exclusively based on open source software and tools. This system ran for about 7 years and was a milestone for the researchers in Chemnitz as well as for the whole HPC community in Germany. The main achievement of the existing CLiC system, the excellent price performance ratio was retained as one of the main goals for the new CHiC system which is described in this article.

The experiences with the aged CLiC set some further goals we had to fulfill. The most error prone components of the old system were the local hard disk, the memory modules and the power supply. On the software side the Andrew Filesystem (AFS) was sometimes difficult to handle and to stabilize. To avoid or at least mitigate the above problems, the new system had to remove or improve these components. Therefore we decided to run diskless compute nodes using server components and ECC protected memory modules. The whole software repository should reside in a high performance clustered file system to enhance the access performance to the application data and avoid some problems of AFS during application runs.
The budget for the new system was set to about 2.6 Mio Euro. For this budget we had to design a balanced machine which would dissipate not more heat than 200 kilowatt. Furthermore, we decided to only self-design the cluster and did not deploy the hardware but we wanted the full responsibility of the software installation process. To support the design process we collected user requirements through project descriptions, questionnaires as well as interviews, and benchmarking of user applications. The results showed that we would need a well balanced general purpose system with high-performance in terms of floating-point operations per second as well as memory bandwidth, and job throughput capabilities. In Figure 1, the concept of the targeted cluster system is shown.

Compute Nodes After an evaluation of current commodity processors at that time we found that the best price-performance ratio could be achieved with dual processor SMP systems. The emerging dual core processors improved this further. The processor of choice could be one of AMD Opteron, Intel Xeon or IBM PowerPC-970. For the migration from the old CLiC to the new system the Intel and

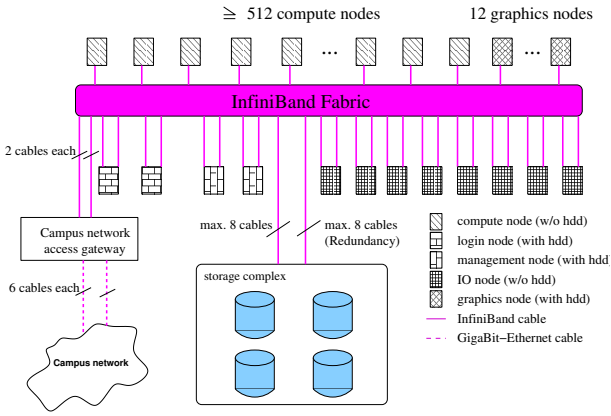

Figure 1. Cluster Concept at a Glance

AMD processors seem to be best suited but we left this decision to the vendors. A compute node had to be equipped with 4 GiB of main memory which results from the user requirements. As mentioned above, to improve the stability of one node we decided against hard disks and use server components with better mean time between failures (MTBF).
To support some special projects, we integrated 12 nodes with graphics card accelerators in the cluster.

Network Due to the higher budget it was possible to design the system with a high-performance network in terms of latency and bandwidth as compared to the communication network of the CLiC which was a Fast Ethernet network comprising one big switch. To simplify the network management we decided to request for proposals using only one network for all tasks like communication, storage I/O, management, monitoring and campus connectivity. The only network architecture which offered capabilities for low latency, high bandwidth, quality of service (QoS), congestion control, combined with a broad range of software APIs was InfiniBand [11, 12]. We had gained experiences with this network technology since its market introduction in 2002 [5, 8, 10, 9, 20, 24]. InfiniBand as a switch-based network technology supported a 288-port switch as the biggest single switch solution at that time. Therefore, we had to plan a hierarchy of InfiniBand switches which is shown in Figure 2. This hierarchy has some advantages like the availability, if one of the big switches fails all nodes could communicate with half the bandwidth (5Gbit/s) in the average case. If one of the small switches fails only 12 nodes would not be reachable. The disadvantage of using this switch hierarchy is that there are communication patterns which could half the bandwidth in the worst case. We could not mathematically prove this but one can intuitively lay these patterns over the hierarchy.

Storage For the new system, there were no special storage requirements regarding the capacity. Therefore, we had

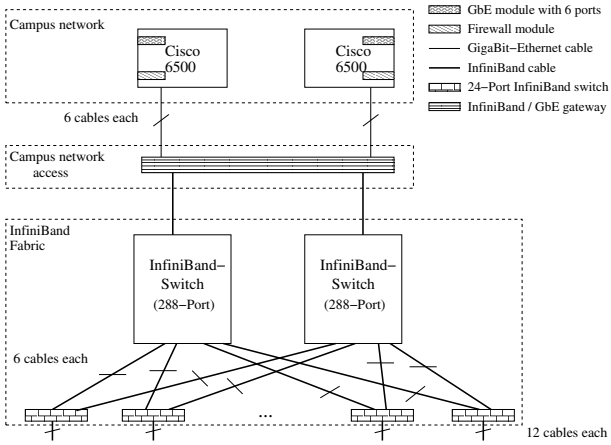

Figure 2. Network Concept at a Glance

taken the latest storage consumption of the project directories on the old system and calculated the necessary capacity for the next 5 years under the assumption that the needed capacity doubles every 12 months. To ensure I/O scalability and a good price-performance ratio we decided to request at least 2GBytes/s of aggregate throughput to the hard disks. The storage complex should be as redundant as possible therefore we required RAID level 6 or better which means that 2 hard drives could fail without data loss. Also the connectivity to the cluster had to be redundant.
On the software side, we chose Lustre from Cluster File Systems Inc. because of its good performance [6], native InfiniBand support, robustness and open source availability. We tested other open source filesystems like PVFS2 and GFS but both didn't show equivalent properties [15]. We left it to the vendors to offer a complete proprietary solution which would be conforming to our requirements.

Software Since we had decided to run diskless compute nodes we looked for a maintained open source toolkit to facilitate this setup. The only software package we found was the Warewulf cluster toolkit [16]. It supports the creation of node images, and the provisioning, management and monitoring of these nodes. As the underlying Linux distribution we chose Scientific Linux 4.x as RedHat Enterprise Linux clone due to its support in the computing center.
All the remaining software for the tasks of monitoring, management, message passing, development and job startup were required to be completely open source. The only exception from this was the procurement of an optimizing compiler suite and math library for the offered hardware architecture.

2.2 Hardware

The CHiC consists of 530 compute, 12 visualization, 8 I/O, 2 management and 2 login nodes. All nodes are connected with a high speed InfiniBand network and connected to a 60 TiB (80 TiB gross) storage complex running the parallel filesystem Lustre. The hardware was delivered by IBM (nodes), Voltaire (InfiniBand interconnection network) and Megware/Xiranet (Storage System) and was installed in 18 water cooled racks from Knürr.

A compute node (IBM x3455) comprises two AMD Opteron 2218 Dual-Core 2.6GHz CPUs, 4GiB DDR2 (667MHz) ECC RAM, a single-port Voltaire InfiniBand 410Ex HCA and an Ethernet port with IPMI support. Each visualization node (IBM IntelliStation A Pro) is equipped with two AMD Opteron 285 Dual-Core 2.6GHz CPUs, 4GiB DDR (400MHz) ECC RAM, a two-ported Voltaire InfiniBand HCA 400 (PCI-X) and an Ethernet port (without IPMI). The visualization nodes are also equipped with an nVidia Quadro FX 4500 X2 graphics card and two 250GiB SATA HDDs. The I/O nodes are identical to the compute nodes except that they have 16GiB DDR2 (667MHz) ECC RAM, a two-ported Voltaire InfiniBand HCA 400Ex and a 80GiB SATA HDD. Two I/O nodes have an integrated LSI SAS controller. A management node (IBM x3755) contains two AMD Opteron 8218 Dual-Core 2.6GHz CPUs, 6GiB DDR2 (667MHz) ECC RAM, 4 Ethernet ports with IPMI support, one two-ported Voltaire InfiniBand HCA 400Ex and a 4x300GB 10k SAS RAID5 with hot-spare. The login nodes (also IBM x3755) are similar to the management nodes with the exception that they have four AMD Opteron 8218 Dual-Core with 2.6GHz and 16GiB DDR2 (667MHz) ECC RAM.

The nodes are connected with four different networks instead of one single InfiniBand network as planned originally. The reason for this was that IBM could sell the x1350 cluster product only with all these networks bundled. Maybe, an InfiniBand-Only installation had been possible with an IBM Business Partner but there was no appropriate offer. Each node connects to the 10Gbit/s InfiniBand fabric, a low-end Gigabit-Ethernet network, a serial console network and a Keyboard-Video-Mouse (KVM) network. The InfiniBand switch components, comprising two 288-port switches (ISR 9288) and 46 24-port switches (ISR 9024S), form a 5-stage Clos network. This network is mainly used for computation and some administration tasks.

The remaining administration tasks are done through the Ethernet network. Each rack is connected to the other racks by only two Gigabit-Ethernet lines and all compute nodes in one rack are connected to one Gigabit-Ethernet switch which offers full bisectional bandwidth.
The other two networks are used for monitoring purposes only. To connect all nodes with the campus network, we use a special InfiniBand-Ethernet gateway device (ISR 9096) which provides the remaining 48 InfiniBand ports.

The 60TiB storage complex consists of 10 RAID con-

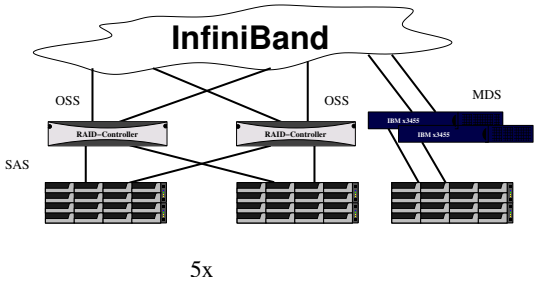

Figure 3. Connection Topology of Storage Complex

troller systems (XAS1000), 10 SATA JBODs[2] with 16 500GiB hard disks each and 1 Serial Attached SCSI (SAS) JBOD with 16 36GiB SAS hard drives. The SAS JBOD is connected to two of the I/O nodes and serves as meta data repository for the parallel filesystem Lustre. For object data storage the 10 SATA JBODs are separated in 20 RAID-6 formations which are managed by 20 RAID controllers (two RAID controllers per host). The connection topology is shown in Figure 3. This Figure also shows the redundant approach by creating pairs of RAID controllers and JBODs.

2.3 Software

Using Scientific Linux 4.4 (RedHat Enterprise Linux clone) ensures the best support for all hardware (especially the IBM x3755 systems) and software components (especially Lustre parallel filesystem) we had installed. Another reason is the usage of this distribution in the local computing center. To further facilitate the installation process we decided to work with the Extreme Cluster Administration Toolkit (xCAT) [3] in conjunction with the Warewulf toolkit to run diskless and diskful nodes under one administration domain. We use Nagios version 2.9 to monitor all the nodes and infrastructure components.

On the system side the Open Fabrics Enterprise Edition InfiniBand software stack in version 1.1 is used. To accelerate the I/O throughput we installed the object based parallel filesystem Lustre 1.6.0beta7 where all home/project directories and software installations reside. This filesystem includes native InfiniBand support and offers high throughput performance.

For development of application codes the GNU compiler suite in version 3.4.6 and 4.2.0 as well as the Qlogic EKOPath Compiler suite 3.0 were installed. As MPI middleware Open MPI 1.2, MVAPICH-0.9.9 and MVAPICH2-0.9.8 can be used. Several math libraries, like Goto BLAS 1.13 and AMD Core Math Library (ACML) 3.6.0 are available for users. To easily manage this software set and their environment variables the Module [4] tool was installed.

To facilitate requests for nodes we installed the resource management system TORQUE in version 2.1.8 and the scheduler Maui in version 3.2.6p20. This ensures a seamless migration from the old system where a similar installation using OpenPBS was used because the user commands are the same.

2.4 Experiences

During the installation and the first months of production, several experiences were gathered on hardware and software level. Generally, the IBM hardware seems to be very stable and reliable so far. However, get the best memory performance a BIOS update was necessary which doubles the achievable memory bandwidth. The management controllers (IPMI) has a documented feature that they are not available during PXE boot stage. Sometimes, after rebooting a node it might happen that the node does not get a DHCP lease. In this case the only way to reboot the node again is to use the switched power distribution unit. Otherwise, the IPMI information is really helpful in finding hardware defects if they occur.

The InfiniBand network is performing really well but some minor drawbacks of the current software installation could be revealed. We tried using IP over InfiniBand in an high-availability mode on our server nodes but when migrating the IP address from one port to the other the server node itself was not available anymore from the other nodes. We are convinced that the problem will be solved with OFED version 1.2. From time to time we experience a similar problem where a random node can not reach the management node but all the others. This might relate to the same problem as described before. On the InfiniBand hardware level, the InfiniBand-to-Ethernet gateway revealed a single point of failure, the software image. To ensure full redundancy and no single point of failure one would have to insist on buying at least two devices. We accepted the one-device-solution with hardware redundancy in the internal fabric due to delivery problems of other solutions.
A documented problem with the InfiniBand stack itself occurs if the system() C-function, which in itself calls fork(), is called. This leads into a failure during the job run and an abort of the job. To solve this problem we have to install a relatively recent vanilla kernel and the latest versions of the InfiniBand stack and MPI implementations.

The Lustre filesystem shows good performance numbers as can be seen in the next section but sporadically it occurs that the Metadata server behaves strangely when the filesystem is under load. Currently, this could be seen when running several stress tests but not with production codes. Here we also believe that an update to the latest stable ver-

[2] JBOD - Just a Bunch of Disks, means here a chassis with special controller hardware

sion might resolve this issue. The biggest drawback of current Lustre implementation is that, if one Object Storage Target – OST[3] fails and is lost, one part of the filesystem is missing. Due to the even distribution of files over all OSTs the loss of one OST could mean that the remaining data is useless and one is required to replay a backup of lost files. That is, the used RAID level should be as redundant as possible to make this problem less likely. Besides these minor issues, the Lustre filesystem exhibits a really good failover capability. The only task to do is to mount the filesystem on the hot-standby metadata or object storage server and ensure connectivity. Due to the several issues we were facing with the Lustre parallel filesystem we are now proposing to have some kind of backup-system.

The batch system TORQUE is adaptable to all problem cases, has a simple configuration and a good support for diskless clients. However, we could not configure all of our policies with the standard configuration process, therefore we have written a wrapper-script to the main user command *qsub* which enforces these policies now. Other goodies are the big user community and a Python interface to the batch system.

3 Benchmarks

For assessing the effectiveness of the cluster system and its software stack we have performed several micro-benchmarks and application runs. This will show the individual and combined performance of several subsystems.

3.1 Synthetic Benchmarks

In the following we will present performance numbers of STREAM, Intel MPI Benchmarks (IMB), High-Performance Linpack (HPL) and Interleaved Or Random (IOR) benchmark.

STREAM The STREAM benchmark [19] is a simple but effective stress test of the memory subsystem. The benchmark consists of four kernels, COPY, SCALE, ADD and TRIAD. TRIAD performs the operation

$$a[i] = b[i] + q \cdot c[i]$$

with vectors of 2 million double precision elements (8 Byte words). This is supposed to avoid cache effects. Furthermore one can simply calculate the achieved floating-point performance. One iteration step of the above calculation includes two floating-point operations. This is multiplied with the number of iterations and then divided by the execution time. Using the result of memory speed and floating-point performance one can calculate the balance of the system which is defined as

$$balance = \frac{peak\ floating\ ops/s}{sustained\ memory\ ops/s}$$

This balance can be interpreted as the number of floating-point operations that can be performed during the time for an average memory access. To calculate the sustained memory ops/s one must simply divide the measured memory bandwidth by the number of bytes of one double precision element, in our case 8 bytes.

To get more comparable performance numbers we benchmarked an Intel Woodcrest system (2.0GHz dual-SMP dual core, 533MHz[4] DDR2 main memory) and one of our compute nodes[5] using several numbers of DIMM modules (2, 4 and 8) in the machines. The gcc-4.2 and the PathScale-3.0 compilers with the `-O3`, `-march/-mcpu/-mtune` flags set to the appropriate architecture were used to compile the benchmark. OpenMP support was enabled and the additional `-fprefetch-loop-arrays` flag was used for for gcc-4.2.

Table 1 shows the results of the TRIAD benchmark, including the measured memory bandwidth and the "balance" as described above. The peak floating-point performance of a single Opteron core is $2FLOP/cycle \cdot 2.6$GHz and $4FLOP/cycle \cdot 2.0$GHz for a single Woodcrest core.

The first observation is the clear advantage of the PathScale compiler for both architectures. It seems that the prefetching of data from memory is much better implemented with this compiler. Another problem we have been facing is the high variance in the results achieved with the `-fprefetch-loop-arrays` optimization flag of gcc compiler running with 4 threads. Here we took the best value for comparison but sometimes the achieved bandwidth is only half of the given values. Finally, one can clearly see the advantage of the AMD architecture with integrated memory controller versus the shared memory controller of the Intel one.

At the time of procurement there were no official Intel compiler available for the AMD64 architecture. More recent benchmarks we made with version 10 of Intel's compiler suite have shown the same relative gap between AMD and Intel processors. Using the latest Intel compilers improved

[3] Lustre differentiate between Object Storage Server (OSS) and Object Storage Target (OST). The latter one represents the real block device and provides access to the chunks of user files. The OSS provides the network request handling for one or more local OSTs

[4] We had only these DIMM modules in the machines but the 667MHz DDR2 modules would only be marginally better

[5] Due to the multiplier used in the Opteron to get the CPU speed, 2.6GHz is a perfect match with the memory speed of 667MHz because no decrease in memory bandwidth is necessary

		gcc-4.2				pathscale-3.0			
		Opteron		Woodcrest		Opteron		Woodcrest	
		BW (MB/s)	Balance	BW (MB/s)	Balance	BW (MB/s)	Balance	BW (MB/s)	Balance
1 Thread	2 DIMMs	3294.2	12.6	3063.7	20.9	5655.7	7.3	3672.8	17.4
	4 DIMMs	3227.1	12.9	3252.0	19.7	5572.9	7.4	3896.4	16.4
	8 DIMMs	3731.0	11.1	3338.1	19.2	5769.8	7.2	3959.6	16.2
2 Threads	2 DIMMs	3708.6	22.4	3230.5	39.6	6056.0	13.7	3967.9	32.2
	4 DIMMs	3212.3	25.9	4345.8	29.4	6114.7	13.6	5061.7	25.3
	8 DIMMs	4854.7	17.1	5232.6	24.5	6520.9	12.7	5876.6	21.8
4 Threads	2 DIMMs	3142.9	52.9	3255.1	78.6	5025.1	33.1	3949.3	64.8
	4 DIMMs	7426.8	22.4	4322.3	59.2	11527.4	14.4	5111.2	50.1
	8 DIMMs	9345.7	17.8	5294.5	48.3	12796.4	13.0	5653.6	45.3

Table 1. Results of STREAM TRIAD Benchmark

the bandwidth to the memory compared with the PathScale compiler results.

IMB The Intel MPI Benchmarks [13] provide a set of concise communication kernels for evaluating the most important MPI functions. It delivers simple timings and throughput values for message sizes between 1 Byte and 4 MiB in the standard mode.

Our goal is to compare the four different MPI implementations, Open MPI 1.2.0, MVAPICH2-0.9.8, MVAPICH-0.9.8 and MVAPICH-0.9.9beta. For space reasons we only compare the PingPong, PingPing, SendRecv, Allreduce, Alltoall and Broadcast benchmark results because they are the most important ones for us.

The Ping-Pong kernel uses the blocking MPI_Send() and MPI_Recv() functions to implement its well-known unidirectional communication pattern. The Ping-Ping kernel uses the non-blocking MPI_Isend() and starts this operations on both sides simultaneously and then block in an appropriate MPI_Recv(). In this way it is similar to a Ping-Pong benchmark with non-optimal conditions (oncoming traffic). To measure the bi-directional performance, the Send-Recv kernel establishes a periodic communication chain where each process receives from the left and sends to the right. This benchmark should reveal the possible full-duplex bandwidth. The collective benchmarks, in our case Allreduce, Bcast and Alltoall, are simple calls to their appropriate collective MPI functions with a simple root-rotation (cf. [13]).

In Figure 4 and 5 we show the measured bandwidth for the PingPong and PingPing benchmark using two nodes (1 core per node). The results for SendRecv, Allreduce, Alltoall and Broadcast using 32 nodes (1 core per node) are shown in Figure 6, 7, 8 and 9.

In the PingPong benchmark all MPIs show nearly the same numbers and achieve bandwidth values of about 900MB/s whereby Open MPI exhibits a little bit better bandwidth for large messages and a bit worse for small mes-

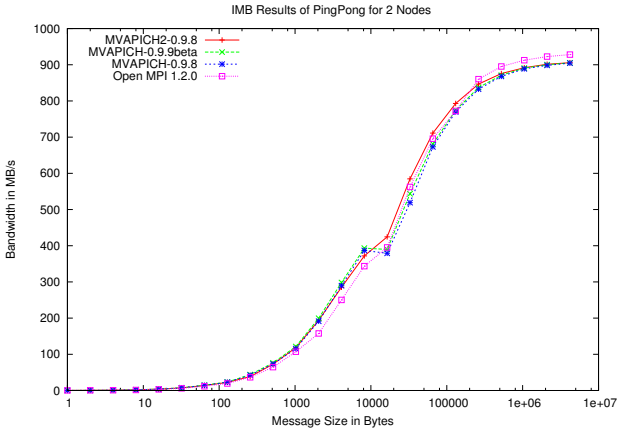

Figure 4. PingPong Results

sages. Between 8KB and 16KB message size all MPIs implements the transition from an eager protocol to a handshake protocol.

Bigger differences among the MPI implementations can be seen in the PingPing benchmark which simulates a non-optimal condition. The maximum bandwidth achievable is 700MB/s for MVAPICH-0.9.8, about 750MB/s (Open MPI) and about 800MB/s (MVAPICH2 and MVAPICH-Beta) when using a 4MB message size.

The SendRecv test shows twice the bandwidth of the one-way PingPing benchmark as expected but a strange behavior can be seen for MVAPICH2 which achieves less than half the bandwidth of the other MPIs. Sometimes this effect is visible on higher node counts but we have no explanation, currently. When running on 2, 4, 8 and sometimes 16 nodes it achieves the same bandwidth as MVAPICH-0.9.9beta. This effect is still under investigation.

Another transition from InfiniBand inline send to "normal" send operations seems to be visible in the Alltoall benchmark as the first buckling. Another buckling for Open MPI is seen again between 8KB and 16KB message size which comes from the protocol transition. Maybe due to some

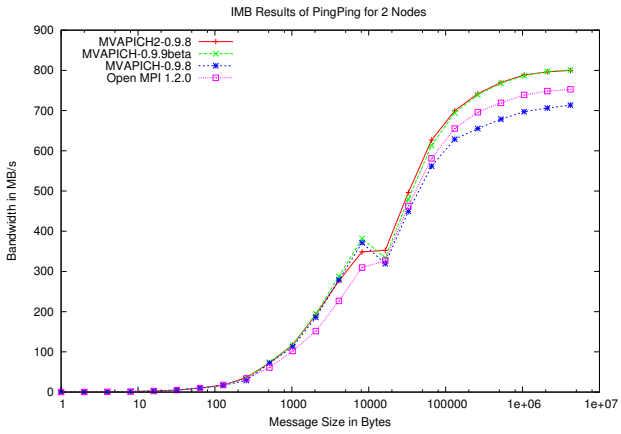

Figure 5. PingPing Results

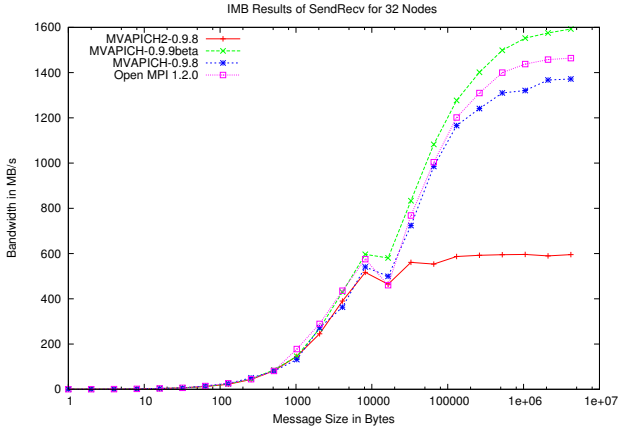

Figure 6. SendRecv Results

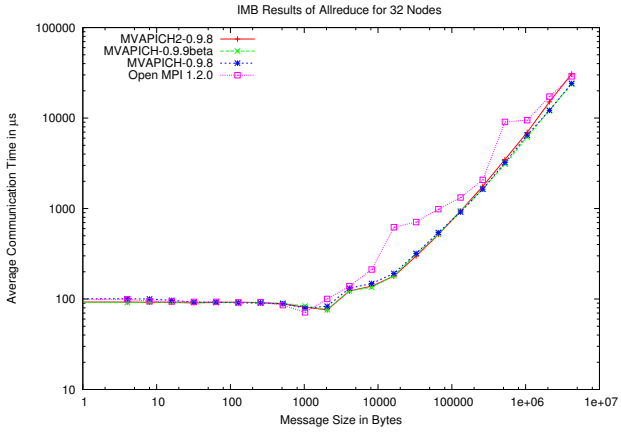

Figure 7. Allreduce Results

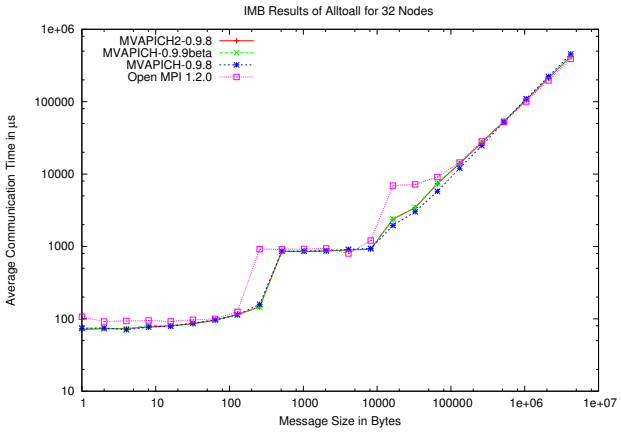

Figure 8. Alltoall Results

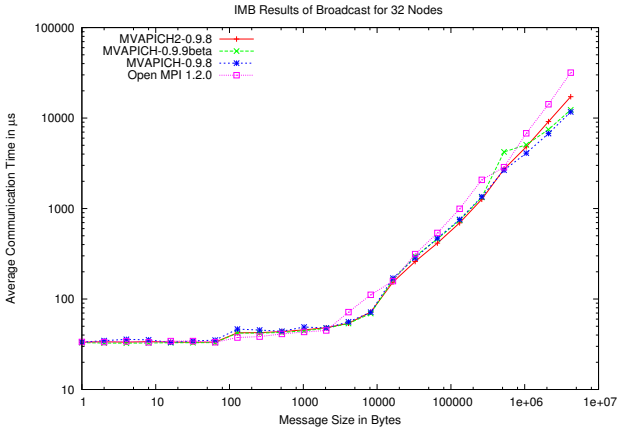

Figure 9. Broadcast Results

optimizations of the Alltoall MPI function the other MPIs didn't show this behavior at this message size. The latency of this MPI collective for small message sizes is worse for Open MPI compared to the other MPIs.

The Allreduce benchmark exhibits a worse behavior for Open MPI compared to the other MPIs as already seen in the Alltoall benchmark. The Broadcast benchmark shows no significant differences among the several MPIs.

The biggest problem with this benchmark is the ambiguous interpretation of the results. For some parameter testing this benchmark seems to be a good test tool but for an overall evaluation of the several MPI implementations it should always be used in combination with application benchmarks. One example is the usage of polling, or callback triggered completion. Polling is the fastest method for waiting on messages but wastes CPU cycles. So, what is good for micro-benchmarks need not necessarily be good for real applications [2].

HPL Solving a system of linear equations is fundamental in the field of scientific computing. The typical way to implement an algorithm to solve such a system of linear equations is using an LU factorization and a backward substitution.
The High-Performance Linpack benchmark [21] solves a random dense linear system on distributed-memory computers using the above methods. The aim is to measure the maximum floating-point performance of a supercomputer. The algorithm itself is scalable but depends slightly on the latency of the communication network and on the memory subsystem which is already shown in Figure 10. For this test we used merely 4 nodes with 16 cores and compared when using 4 or 8 memory slots out of 12, and using the TCP stack with IP over InfiniBand or the native InfiniBand verbs inside the MPI implementation. The parameters were always the same in the input file for the benchmark and we measured the floating point performance for several process grids. Using the definition of the efficiency below we can achieve an about 3% better efficiency when using the native InfiniBand verbs which exhibits a much lower latency. We can add a further 1% if we use 8 instead of 4 memory slots. We repeated the benchmark several times and the relative gap was still the same. Running on 4 nodes we could achieve an efficiency of 84% when taking the best measured value into account.

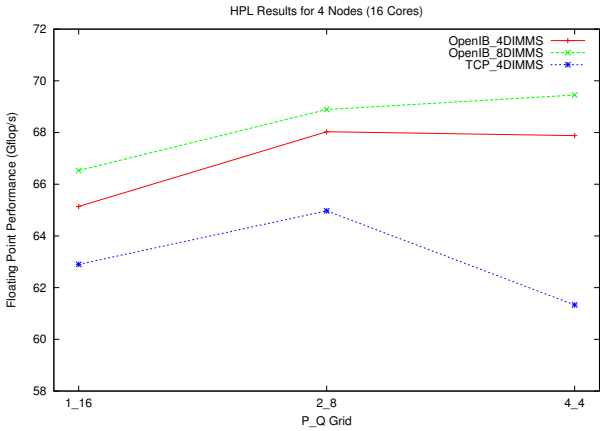

Figure 10. HPL Results

Therefore, this benchmark can be used to get another measure of the balance of the whole system, the efficiency.

$$system\ efficiency = \frac{Rmax}{Rpeak}$$

Rmax is the measured HPL performance and *Rpeak* is the theoretical peak performance which is presented in the STREAM paragraph. The result of this benchmark is used for the well-known bi-annual Top500 list of the fastest supercomputers in the world on which this benchmark was run. The problem with this benchmark is that it primarily assess one aspect of today's supercomputers mainly, the floating-point performance. A ranking depending only on this result is not expressive enough to assess a supercomputer. To overcome this problem the HPC Challenge suite [18] was composed.

The biggest measure we have finished on the CHiC was using 520 nodes (2080 cores), the PathScale-2.4 compiler suite, MVAPICH-0.9.7-mlx2.2.0 (shipped with OFED-1.1) and the Goto-BLAS library version 1.10. The achieved result was 8210 GFlop/s which is a 76% efficiency value. The CHiC was entered in the Top500 list with rank 117 in June 2007 (rank 237 in November 2007) . Using an equivalent system utilizing Intel Woodcrest CPUs one could achieve about twice as much.

IOR The best way to assess the performance of the parallel file system and its underlying hardware components is to benchmark with several application access patterns. The benchmark b_eff_io [23] is aimed at producing a characteristic average number of the I/O bandwidth achievable with parallel MPI-I/O applications exhibiting various access patterns. The result should be a comparable number for storage systems similar to the Top500 benchmark. This benchmark was not executable on our Lustre file system due to lack of full POSIX locking support in version 1.6.x of Lustre. Therefore we have chosen another benchmark, IOR [6], which fullfills the above requirement. IOR is a parallel file system bandwidth testing code which was initially developed to test GPFS [25] from IBM on ASCI Blue Pacific and White machines at the Lawrence Livermore National Laboratory [28]. The supported access patterns were an attempt to represent ASC application's access patterns.

The benchmark has the capability of 3 access patterns, "one file per process", "shared file segmented access", and "shared file strided access". The main difference of the two shared file access patterns is whether the data of a process is contiguous (segmented) or non-contiguous (strided) in the file. Several interfaces like POSIX and MPI-IO are available with the possibility to fine-tune some interface specific parameters like the usage of collective functions with the MPI-IO interface. The result of the benchmark is always the best read/write bandwidth achieved among all repetitions. The implementers justify this with the argument that they run the benchmark during the production cycles where other applications access the storage system simultaneously. Our benchmark runs were performed during the production cycles as well.

In Figure 11 and 12 we show the read and write performance for the three access patterns described above using the POSIX interface on several node counts. For the "file per process" case we measured with striping of the 2.5GB file over one object storage target (OST) or 20 OSTs. For

the "shared file" test cases the single file is always striped over 20 OSTs. The file size is $No.of Nodes \cdot 2.5GB$ whereby each node reads/writes a 2.5GB data set in this case. The transfer size parameter of IOR is set to 1MB which is the stripe size of the Lustre file system installation.

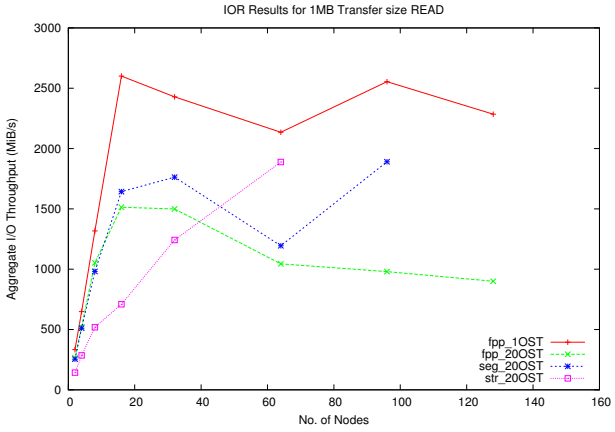

Figure 11. IOR Read Results

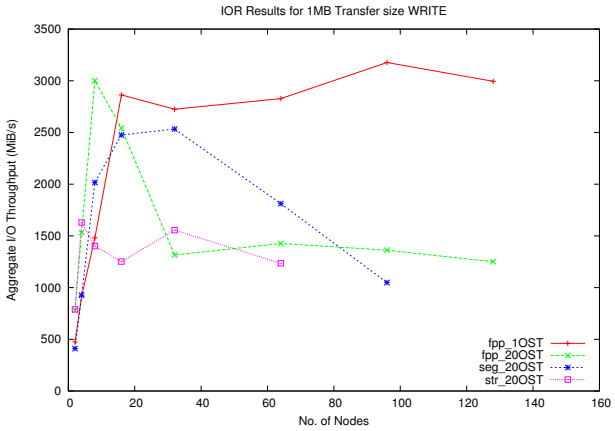

Figure 12. IOR Write Results

The highest values could be achieved with the "file per process" test case with no file striping running on 96 nodes, 3.2 GiB/s write and 2.6 GiB/s read performance. These results can be held relatively stable if at least 16 nodes are working on a big file per process. This comes from the separation of files among the OSTs. In this case using 16 nodes each file is put on a separate OST. This means also a relatively good scalability of this access pattern up to 500 nodes.
If a file is striped over all 20 OSTs the concurrency of accessing the hard drives shows a major impact on performance results for the "file per process" test case when using more than 16 nodes. If less than 16 nodes are used the write performance is much better compared to the no striping case. The read performance for this test case is always worse due to much higher seek time overhead.
In the "shared file" benchmark the performance numbers for node counts of 96 and 128 could not always be measured due to some strange behavior of the Metadata server during the runs. We believe that the reason is the usage of the 1.6Beta7 version of Lustre. The strided case shows for node counts of more than 8 nodes a bad write performance which was expected due to the non-contiguous access pattern. The segmented case shows nearly the same write performance as the "file per process" case on small node counts which was also expected since the segmented access pattern is nearly equivalent to the "file per process" one using one or several OSTs. For higher node counts the access pattern corresponds more with "file per process" case using 20 OSTs which can also be seen in the performance numbers.
The biggest influences on the read/write performance when striping over all 20 OSTs is the number of locks which are necessary to access the part of the file, and the slow seek time of the SATA disks. More performance with the same number of storage servers can be gained by using more hard drives and thus more OSTs per server when running with a big number of clients.

3.2 Application Benchmarks

In this section we are presenting some performance numbers of real world applications, which are used at our site, that we could gather during the tender process. These application runs were done on a 16 node Intel Woodcrest cluster and a 16 node AMD Opteron cluster. Both used InfiniBand as interconnect and the nodes were dual processor dual core machines. The Intel cluster comprised 3.0GHz CPUs, 8GB RAM per node with Intel compiler suite 9.x and math kernel library 8.x installed. The nodes of the AMD system were similar to the current CHiC nodes except that they comprised 8GB RAM per node with all memory slots filled. This system had installed the PathScale compiler suite 2.3 and the appropriate AMD math core library 3.0. For benchmarking, both compilers were used with no aggressive optimization settings.

ABINIT ABINIT is a package for quantum mechanics calculations whose main program allows one to find the total energy, charge density and electronic structure of systems made of electrons and nuclei (molecules and periodic solids) within Density Functional Theory (DFT), using pseudo-potentials and a planewave basis. The results for a small $Si - SiO_2$ system [7] with 43 atoms, 126 bands, 48728 plane waves and a 61x61x256 FFT grid is shown in Table 2. In this benchmark the Opteron system shows a 5% advantage compared to the Woodcrest system. This result seems to be mainly influenced by the speed of the memory subsystem.

	AMD Cluster	Intel Cluster
Time in s	1,384.6	1,454.2

Table 2. Results of ABINIT Benchmark on 32 Cores

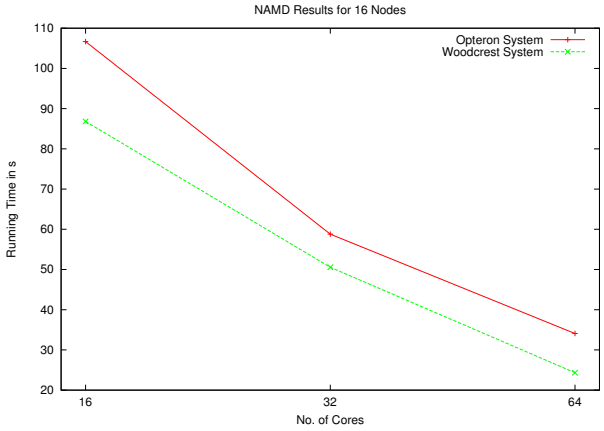

Figure 13. Results of ApoA1 Benchmark

NAMD NAMD is a parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems. Based on Charm++ parallel objects [14] which provide adaptive overlap of communication and computation across modules, NAMD scales to hundreds of processors [26]. For benchmarking we used the ApoA1 test case [22] which calculates a complex system of 92,224 atoms and is therefore a good estimate of performance for a long production simulation. We benchmarked this test case on 16 nodes with a various number of processor cores as shown in Figure 13. One can clearly see the good scaling behavior of the application when adding more cores per node. This also means that the memory subsystem plays no primary role. Due to its overlap of communication and computation there is also no major impact by the MPI implementation. Finally, the computation throughput of the processor is the primary accelerator and thus, the Intel Woodcrest exhibits the best results.

4 Conclusions

In this paper we presented our design decisions for a 2048 processor core cluster using the InfiniBand high-speed interconnect and the Lustre parallel filesystem. We showed that finding a balanced system for a limited budget is a challenging task.

We presented benchmark results using micro-benchmarks and real world applications. With the STREAM memory bandwidth benchmark the AMD Opteron can outperform an Intel Woodcrest system by a factor of 2. Taking the HPL (maximum floating-point performance) into account than it is exactly reverse. By comparing the running time of two application test cases we also got no real winner. The answer which architecture is suited or not is, it depends. For the system we purchased the accumulated benchmark results were almost similar between Intel and AMD architectures. We chose IBM because they offered the better overall system approach.

The Lustre parallel filesystem over our storage system exhibits 3.2 GiB/s write and 2.6 GiB/s read bandwidth measured with IOR and 96 nodes. Under load conditions it happens from time to time that the metadata server shows some stability issues but we believe, with installing the latest Lustre version and running with the latest OFED stack these issues will disappear.

Figure 14. View on the CHiC (1 row)

5 Future Work

To further enhance the performance of the cluster we will work on several software components of the system. First of all we are trying to shrink the node image to only 50MB from currently 300MB to increase the available application memory. For better configurability we are planning a new qsub command implementation using the Python interface. To accelerate the I/O speed to the Lustre parallel filesystem we have the intention to create some kind of hierarchical storage management. Therefore we are investigating the usage of a Lustre filesystem in a RAM disk. Another small project will be the usage of our graphic accelerator cards in the 12 visualization nodes for scientific computing. Our work on MPI implementations and InfiniBand which is one of the topics of our research group will bring further optimization to the system.

References

[1] Forschung mit Profil. *[http://archiv.tu-chemnitz.de/pub/2005/0148/data/05_TU_SonderheftBS.pdf]*, ISSN 0946–1817.

[2] R. Dimitrov and A. Skjellum. Impact of Latency on Applications' Performance. In *Proceedings of the Fourth MPI Developer's and User's Conference*, March 2000.

[3] E. Ford, B. Elkin, S. Denham, B. Khoo, M. Bohnsack, C. Turcksin, and L. Ferreira. IBM Redbook, ISBN 0738426776, 2002.

[4] J. L. Furlani and P. W. Osel. Abstract Yourself with Modules. In *Proceedings of the Tenth Large Installation Systems Administration Conference (LISA '96)*, 1996.

[5] R. Grabner, F. Mietke, and W. Rehm. An MPICH2 Channel Device Implementation over VAPI on InfiniBand. In *Proceedings of Workshop on Communication Architecture for Clusters (CAC'04) held in conjuncation with IPDPS*, 2004.

[6] R. Hedges, B. Loewe, T. McLarty, and C. Morrone. Parallel File System Testing for the Lunatic Fringe: the Care and Feeding of Restless I/O Power Users. In *Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2005)*. IEEE Computer Society, 2005.

[7] T. Hoefler, R. Janisch, and W. Rehm. Parallel scaling of Teter's minimization for Ab Initio calculations. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC06)*, 2006.

[8] T. Hoefler, T. Mehlan, F. Mietke, and W. Rehm. Fast Barrier Synchronization for InfiniBand. In *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 4 2006.

[9] T. Hoefler and W. Rehm. A Communication Model for Small Messages with InfiniBand. In *Proceedings of the Parallel-Algorithmen, -Rechnerstrukturen und -Systemsoftware (PARS) Workshop 2005*. ISSN 0177-0454.

[10] T. Hoefler, C. Siebert, and W. Rehm. A practically constant-time MPI Broadcast Algorithm for large-scale InfiniBand Clusters with Multicast. In *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium*, page 232. IEEE Computer Society, 03 2007.

[11] InfiniBand Trade Association. *InfiniBand Architecture Specification Release, 1.2*, volume 1 edition, October 2004.

[12] InfiniBand Trade Association. *InfiniBand Architecture Specification Release, 1.2*, volume 2 edition, October 2006.

[13] Intel GmbH. *Intel MPI Benchmarks, Users Guide and Methodology Description*. Intel GmbH, D-50321 Brühl, Germany, 2006.

[14] L. V. Kale and S. Krishnan. *Charm++: Parallel Programming with message-driven objects*. MIT Press, 1996.

[15] M. Knapp. Evaluierung Paralleler Dateisysteme unter Linux, 2006. Seminar Paper, Computer Architecture Group, Chemnitz University of Technology.

[16] G. M. Kurtzer. Warewulf: The Cluster Node Management Solution. Technical Presentation at Supercomputing Conference 2003 in Phoenix, *http://scs.lbl.gov/html/reports/warewulf-SC2003.pdf*.

[17] R. W. Lucke. *Building Clustered Linux Systems*. Pearson Education Inc., Upper Saddle River, New Jersey 07458, 2004.

[18] P. Luszczek, J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi. Introduction to the HPC Challenge Benchmark Suite, March 2005. *[http://icl.cs.utk.edu/hpcc]*.

[19] J. D. McCalpin. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, December 1995.

[20] F. Mietke, D. Dunger, T. Mehlan, T. Hoefler, and W. Rehm. A native InfiniBand Transporter for MySQL Cluster. In *Proceedings of the 2nd Workshop Kommunikation in Clusterrechnern und Clusterverbundsystemen (KiCC'07)*, 2007.

[21] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, January 2004. version 1.0a, *[http://www.netlib.org/benchmark/hpl]*.

[22] J. C. Phillips, W. Wriggers, Z. Li, A. Jonas, and K. Schulten. Predicting the structure of apolipoprotein A-I in reconstituted high density lipoprotein disks. *Biophysical Journal, 73*, 1997.

[23] R. Rabenseifner, A. E. Koniges, J.-P. Prost, and R. Hedges. The Parallel Effective I/O Bandwidth Benchmark: b_eff_io. *Calculateurs Parallles Journal on Parallel I/O for Cluster Computing, Special Issue*, February 2004.

[24] R. Rex, F. Mietke, C. Raisch, H.-N. Nguyen, and W. Rehm. Improving Communication Performance on InfiniBand by Using Efficient Data Placement Strategies. In *Proceedings, International Conference on Cluster Computing (Cluster 2006)*, 2006.

[25] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *Proceedings of the First Conference on File and Storage Technologies (FAST)*, pages 231–244, 2002.

[26] K. Schulten, J. C. Phillips, L. V. Kale, and A. Bhatele. *Biomolecular modelling in the era of petascale computing*. Chapman and Hall/CRC Press, Taylor and Francis Group, New York, 2008. In press.

[27] J. D. Sloan. *High Performance Linux Clusters with OSCAR, Rocks, openMosix, and MPI*. O'Reilly Media Inc., Sebastopol, California 95472, 2005.

[28] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, and D. D. E. Long. File System Workload Analysis for Large Scale Scientific Computing Applications. In *Proceedings of the 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2004)*. IEEE Computer Society, 2004.