Performance Evaluation of I/O Traffic and Placement of I/O Nodes on a High Performance Network*

Salvador Coll*†, Fabrizio Petrini*, Eitan Frachtenberg* and Adolfy Hoisie*

* CCS-3 Modeling, Algorithms, & Informatics
Computer & Computational Sciences Division
Los Alamos National Laboratory
† Electronic Engineering Department
Technical University of Valencia
{scoll,fabrizio,eitanf,hoisie}@lanl.gov

Abstract

A common trend in the design of large-scale clusters is to use a high-performance data network to integrate the processing nodes in a single parallel computer. In these systems the performance of the interconnect can be a limiting factor for the input/output (I/O), which is traditionally bottlenecked by the disk bandwidth. In this paper we present an experimental analysis on a 64-node AlphaServer cluster based on the Quadrics network (QsNET) of the behavior of the interconnect under I/O traffic, and the influence of the placement of the I/O servers on the overall performance. The effects of using dedicated I/O nodes or overlapping I/O and computation on the I/O nodes are also analyzed. In addition, we evaluate how background I/O traffic interferes with other parallel applications running concurrently. Our experimental results show that a correct placement of the I/O servers can provide up to 20% increase in the available I/O bandwidth. Moreover, some important guidelines for applications and I/O servers mapping on large-scale clusters are given.

Keywords: Interconnection Networks, Performance Evaluation, User-level Communication, Input/Output.

1 Introduction

Scientific applications that run on parallel systems usually require input and output of large amounts of data, therefore the I/O performance can be a potential bottleneck. Collective I/O, in which all processes cooperate to carry out large-scale I/O transactions, has been proposed as a way to improve the I/O performance of such applications. Some techniques currently being used to provide collective I/O facilities are: (1) parallel file systems (HFS for the HP Exemplar [2], PFS for the Intel Paragon [9], PIOFS and GPFS for the IBM SP [5], XFS for the SGI Origin2000 [20], PVFS [3] for Linux clusters), (2) distributed file systems (NFS [21], GFS [17]) and (3) run-time I/O libraries (MPI-IO [8]). Most of these systems assume that the I/O subsystem is homogeneous and the message passing over the network is fast and scalable. Nevertheless, the behavior of the interconnect in such systems can be also a performance limiting factor, although the I/O performance on massively parallel processors has been traditionally limited by disk bandwidth [4].

The efficient integration of the interconnection network with the I/O is a key factor to efficiently exploit the power of high-performance parallel computers. InfiniBand [1] is an emerging standard that provides an integrated view of computing, networking and storage technologies. The InfiniBand architecture is based on a switch interconnect technology with high speed point-to-point links and offers support for Quality of Service (QoS), fault-tolerance, remote direct memory access, etc., and is likely to become the backbone of future commodity parallel computers, I/O servers, and data centers.

The Quadrics interconnection network (QsNET) [7] is currently being used in some of the largest parallel systems in the world, typically connecting Compaq Alpha-based servers, but increasingly other compute platforms too[1]. The QsNET provides some innovative design issues very similar to those defined by the InfiniBand specification, which are likely to appear in the commodity market in the next few years. Some of these salient aspects are the integration of the local virtual memory into a distributed virtual shared memory, remote direct memory access, the presence of a programmable processor in the network interface that allows the implementation of intelligent communication protocols, and fault-tolerance.

[1]More information on the Quadrics network can be found at http://www.c3.lanl.gov/ ~fabrizio/quadrics.html

In [13] we analyzed the QsNET performance under specific load conditions to obtain the "peak performance" of the network and a baseline for further studies. Since not only the efficient support for computation-related traffic patterns but a good integration with the I/O subsystem is a key issue to provide a high-performance platform for scientific applications, in this paper we address the experimental evaluation of the networking and I/O integration in the Quadrics interconnect.

For this reason, we present an analysis of the network behavior under I/O-related traffic patterns and the effect of the placement of the I/O nodes[2] in the cluster. It is shown that the placement of the I/O servers greatly conditions the behavior of the interconnect. Moreover the effect of using either dedicated I/O nodes or I/O nodes that run compute jobs too is analyzed. These experiments are complemented with an evaluation of the interference between the I/O traffic and other simultaneously running applications.

The test bed for the network evaluation was a 64-node QsNET-based AlphaServer ES40 cluster. The performance effects in the network are explained in terms of hardware parameters, flow control and congestion resolution. The results of the analysis in this work provide a complete characterization of the interconnect when routing I/O traffic and have powerful and direct practical implications, for example related to the I/O node mapping and application distribution. The study of the hardware and software primitives used to implement multicast communication and the impact of the network performance on the user applications are outside the scope of the paper and can be found in [12] and [10], respectively.

In the next section we summarize the main characteristics of the network. In Section 3 the experimental methodology is described, while the performance results are presented and discussed in Section 4. Finally, some concluding remarks are drawn in Section 5.

2 The QsNET

The QsNET is based on two building blocks, a programmable network interface called Elan [18] and a low-latency high-bandwidth communication switch called Elite [19]. The network has several layers of communication libraries which provide trade-offs between performance and ease of use.

2.1 Elan

The internal functional structure of the Elan[3] centers around two primary processing engines: the microcode processor and the thread processor.

The 32-bit microcode processor supports four separate threads of execution, where each thread can independently issue pipelined memory requests to the memory system. Up to eight requests can be outstanding at any given time. The scheduling for the microcode processor is lightweight, enabling a thread to wake up, schedule a new memory access on the result of a previous memory access, and then go back to sleep in as few as two system-clock cycles.

The four microcode threads are described below: (1) *inputter thread:* Handles input transactions from the network. (2) *DMA thread:* Generates DMA packets to be written to the network, prioritizes outstanding DMAs, and time-slices large DMAs so that small DMAs are not adversely blocked. (3) *processor-scheduling thread:* Prioritizes and controls the scheduling and descheduling of the thread processor. (4) *command-processor thread:* Handles operations requested by the host processor at user level.

The thread processor is a 32-bit RISC processor used to aid the implementation of higher-level messaging libraries without explicit intervention from the main CPU. In order to better support such an implementation, the thread processor's instruction set was augmented with extra instructions that construct network packets, manipulate events, efficiently schedule threads, and block-save and restore a thread's state when scheduling.

The Elan contains routing tables that translate every virtual process number into a sequence of tags that determine the network route. Several routing tables can be loaded in order to have different routing strategies. Each link provides buffer space for two virtual channels with a 128-entry, 16-bit FIFO RAM for flow control.

2.2 Elite

The other building block of the QsNET is the Elite switch. The Elite provides the following features: (1) 8 bidirectional links supporting two virtual channels in each direction, (2) an internal 16×8 full crossbar switch[4], (3) a nominal transmission bandwidth of 400 MB/s on each link direction and a flow through latency of 35 ns, (4) packet error detection and recovery,

---

[2]I/O node and I/O server will be used interchangeably through the paper.

[3]This paper refers to the Elan3 version of the Elan. We will use Elan and Elan3 interchangeably throughout the paper.

[4]The crossbar has two input ports for each input link, to accommodate the two virtual channels.
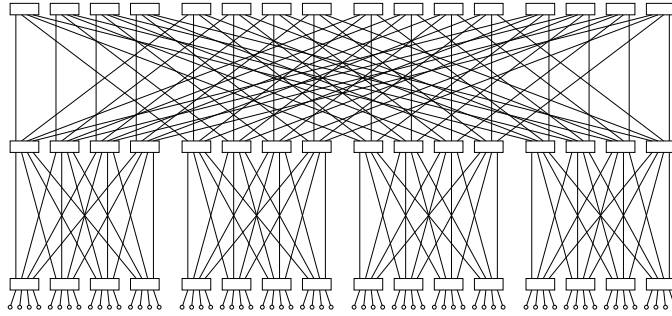
Figure 1. 4-**ary fat-tree of dimension 3**

with routing and data transactions CRC protected, (5) two priority levels combined with an aging mechanism to ensure a fair delivery of packets in the same priority level, (6) hardware support for broadcasts, (7) and adaptive routing.

The Elite switches are interconnected in a quaternary fat-tree topology [11], which belongs to the more general class of the k-ary n-trees [15] [14]. A quaternary fat-tree of dimension n is composed of 4^n processing nodes and $n * 4^{n-1}$ switches interconnected as a delta network, and can be recursively built by connecting 4 quaternary fat trees of dimension $n-1$. A quaternary fat tree of dimension 3 is shown in Figure 1.

2.2.1 Packet Routing and Flow Control

Each user- and system-level message is chunked in a sequence of packets by the Elan. An Elan packet contains three main components. The packet starts with the (1) routing information, that determines how the packet will reach the destination. This information is followed by (2) one or more transactions consisting of some header information, a remote memory address, the context identifier and a chunk of data, which can be up to 64 bytes in the current implementation. The packet is terminated by (3) an end of packet (EOP) token.

Transactions fall into two categories: write block transactions and non-write block transactions.

The purpose of a write block transaction is to write a block of data from the source node to the destination node, using the destination address contained in the transaction immediately before the data. A DMA operation is implemented as a sequence of write block transactions, partitioned into one or more packets (a packet normally contains 5 write block transactions of 64 bytes each, for a total of 320 bytes of data payload per packet).

The non-write block transactions implement a family of relatively low level communication and synchronization primitives. For example, non-write block transactions can atomically perform remote test-and-write or fetch-and-add and return the result of the remote operation to the source, and can be used as building blocks for more sophisticated distributed algorithms.

Elite networks are source routed. The routing information is attached to the header before injecting the packet into the network and is composed by a sequence of Elite link tags. As the packet moves inside the network, each Elite removes the first routing tag from the header, and forwards the packet to the next Elite in the route or to the final destination. The routing tag can identify either a single output link or a group of adjacent links.

The transmission of each packet is pipelined into the network using wormhole flow control. At link level, each packet is partitioned in smaller units called flits (flow control digits) [6] of 16 bits. The header flit opens a circuit between source and destination, and this path stays in place until the destination sends an acknowledgment to the source.

Minimal routing between any pair nodes can be accomplished by sending the message to one of the nearest common ancestors and from there to the destination. That is, each packet experiences two routing phases, an adaptive ascending phase to get to a nearest common ancestor, followed by a deterministic descending phase. The Elite switches can adaptively route a packet picking the least loaded link.

3 Experimental Framework

We tested the main features of the QsNET on a 64-node cluster of Compaq AlphaServer ES40s, running True64 Unix. Each AlphaServer node is equipped with 4 Alpha 667MHz 21264 processors, 8 GB of SDRAM and two 64-bit, 33MHz PCI I/O buses. The Elan3 QM-400 card is attached to one of these buses and links the SMP to a quaternary fat tree of dimension three, as the one shown in Figure 1.

Unless otherwise stated, the communication buffers are allocated in Elan memory in order to isolate I/O bus-related performance limitations, except for the ping tests, whose goal is to provide basic performance results that are a reference point for the following experiments.

3.1 Unidirectional Ping

We analyze the latency and bandwidth of the network by sending messages of increasing sizes. In order to identify different bottlenecks, the communication buffers are placed either in main or in Elan memory, using the allocation mechanisms provided by the lowest level Elan programming library, Elan3lib.

At Elan3lib level the latency is measured as the elapsed time between the posting of the remote DMA request and the notification of the successful completion at the destination. The unidirectional ping tests for MPI are implemented using matching pairs of blocking sends and receives.

3.2 Bidirectional Ping

The unidirectional ping experiments can be considered as the "peak performance" of the network. By sending packets in both directions along the same network path we can expose several types of bottlenecks.

For example, the Elan microcode interleaves four activities, DMA engine, inputter, command processor and thread processor. This test can evaluate how the DMA engine and the inputter can work with bidirectional traffic. Also the link-level flow control requires the transmission of control information, which can lead to a degradation of the unidirectional performance in the presence of bidirectional traffic. Bidirectional traffic is typically generated by permutation patterns in which a node is both source and destination.

3.3 Hot-spot

Under hot-spot traffic, a set of communication partners try to read from or write into the same memory block. This experiment models the behavior of a single I/O server being accessed by multiple clients and provides basic results for better understanding the network. This localized communication pattern can lead to a severe form of congestion known as *tree saturation* [16], which can seriously degrade the overall performance of the interconnect.

3.4 Multiple Hot-spots

The network traffic generated by a parallel job that is performing input/output can be modeled with a collection of hot-spots, where each hot-spot is a node that acts as an I/O server[5] and is the target of multiple messages originated by the other nodes. This test has been designed to analyze the behavior of the network when one parallel job is performing transactions over several hot nodes. We address five distinct performance dimensions.

1. I/O read/write ratio: the ratio between I/O reads and writes is a number between 0 and 1, with 0 being all reads and 1 all writes.

2. The inter-arrival time between two I/O messages issued by a client node can be either uniformly or exponentially distributed.

3. I/O traffic: this parameter defines the access pattern to the I/O nodes. Two patterns have been analyzed:

    (a) uniform I/O, each node performing I/O randomly selects its destination for every transaction and

    (b) fixed I/O, each node uses a fixed destination for all its transactions.

4. I/O node mapping (hot-node mapping): this parameter defines the placement of the I/O nodes in the cluster. Two alternatives have been tested:

    (a) clustered, I/O nodes located in consecutive nodes at the higher nodes locations and

    (b) distributed, I/O nodes uniformly distributed through the cluster.

5. Application mapping: defines whether the application performing I/O runs on the I/O nodes (all nodes mapping) or not (non-I/O nodes mapping).
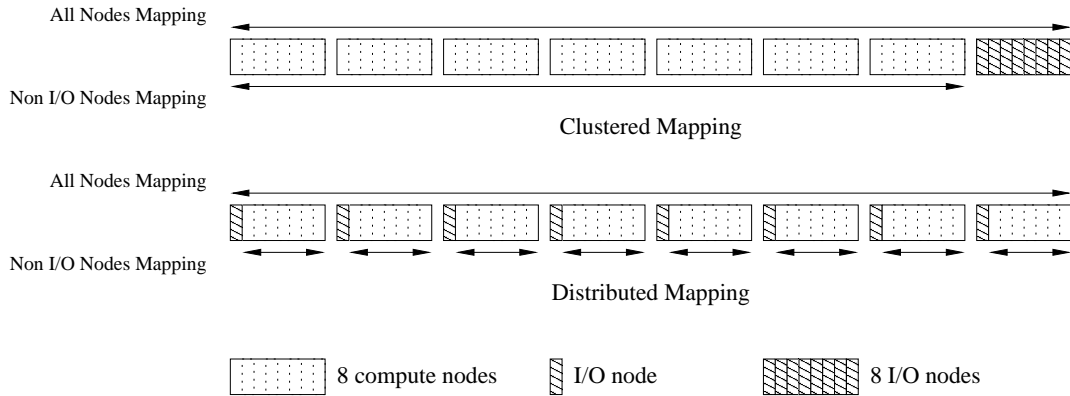
Figure 2. I/O and Application Mappings with 64 nodes and 8 I/O nodes. The arrows highlight the nodes that inject I/O traffic into the network.

Figure 2 describes the types of I/O and application mappings used in the experiments with a configuration of 64 nodes. In the clustered mapping, 8 I/O nodes are placed in the upper part of the network (the dark ones), while the remaining 56 nodes are dedicated to computation. For this type of I/O mapping, we consider two types of application mappings. In the "All Nodes Mapping" all 64 nodes generate I/O traffic. In this case, the I/O nodes are both source and destination of the I/O traffic. In the "Non I/O Nodes Mapping", only the first 56 nodes inject I/O traffic into the network. The I/O nodes are only sinks of the I/O traffic, as outlined by the arrow below the first row.

With distributed I/O mapping, shown in the second row of Figure 2, the I/O nodes are scattered with a stride of 8 over all the nodes. In this case we have an I/O node every 8 nodes. As in the clustered approach, we distinguish the two cases where the I/O nodes do and do not inject messages into the network.

3.5 Combined Traffic

With this benchmark we study how a parallel job that is executing I/O traffic can affect the communication performance of another parallel job that is running concurrently. We perform the tests by running two parallel jobs in the cluster, each one using half of the available nodes. The I/O job generates traffic as described in Section 3.4, while the compute job injects uniform traffic. We analyze the Cartesian product of several performance dimensions, which are outlined in Figure 3. In particular,

1. I/O node mapping: we consider clustered (the upper row of Figure 3) and distributed mapping (the lower row of Figure 3).

2. Application mapping: the two applications running in this test use half of the available nodes with two different approaches. If "All Nodes Mapping" is used each application is mapped to 32 nodes (even if it overlaps with I/O servers). On the other hand, with "Non-I/O Nodes Mapping" the I/O nodes are dedicated servers and each application is mapped to 28 nodes. When we have the all nodes mapping with the clustered I/O, we can distinguish two further cases. The job performing I/O is mapped onto the I/O nodes. As shown in the upper row of Figure 3, this implies that the compute job is mapped on the first half of the network while the I/O job is mapped on the second half. And the symmetric case, where the compute job is mapped onto the I/O nodes.

3. I/O load: the I/O job can inject messages into the network with increasing load.

4 Experimental Results

4.1 Unidirectional Ping

Figure 4 a) shows the performance of the unidirectional ping. The peak bandwidth of 335 MB/s is reached when both source and destination buffers are placed in the Elan memory. The maximum amount of data payload that can be sent by the

---

[5]For this reason in the rest of the paper multiple hot-spots and I/O traffic are used interchangeably.
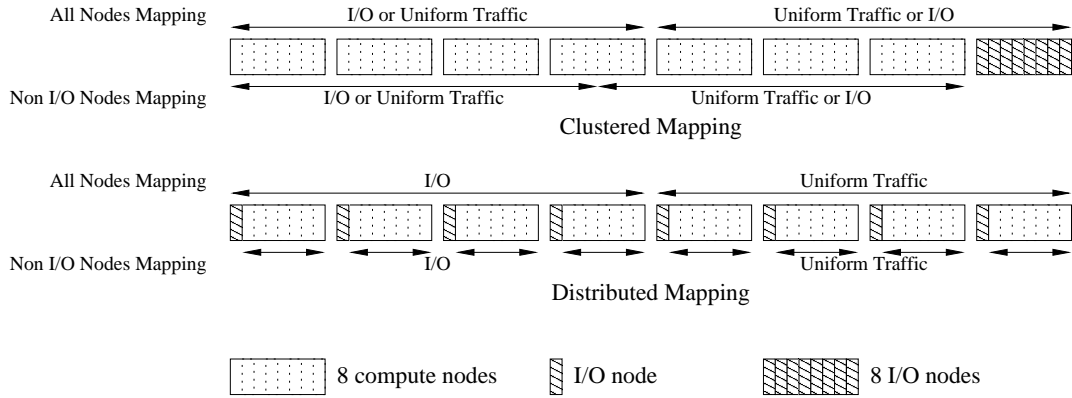
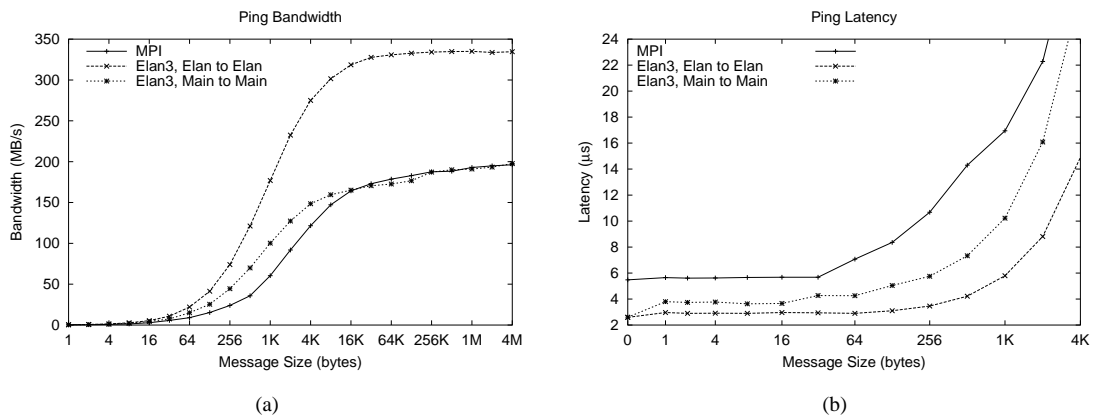Figure 3. I/O and Application Mappings with 64 nodes and 8 I/O nodes with combined traffic.

(a)   (b)

Figure 4. Unidirectional Ping

current Elan implementation in a packet is 320 bytes, partitioned in five low-level write-block transactions of 64 bytes. For this packet format, the overhead is 58 bytes, for the message header, CRCs, routing info, etc. This implies that the delivered peak bandwidth is approximately 396 MB/s, or 99% of the nominal bandwidth (400 MB/s).

But the asymptotic bandwidth for main memory to main memory communication is only 200 MB/s for both Elan3lib (lowest level programming library) and MPI. These results show that the PCI interface running at 33MHz is the bottleneck for this type of communication.

Figure 4 b) shows the latency in the range $[0 \ldots 4KB]$. With Elan3lib the basic latency for 0-byte messages is only 2.2 μs and is almost constant at 2.4 μs for messages up to 64 bytes. We note an increase in the latency at MPI level, compared to the latency at the Elan3lib level, from approximately 2 μs to 5.5 μs. While at Elan3lib level the latency is mostly hardware, MPI needs to run a thread in the Elan microprocessor in order to match the message tags: this introduces the extra overhead responsible for the higher latency.

4.2 Bidirectional Ping

Figure 5 a) shows how the bidirectional bandwidth is degraded by the PCI bus. When the communication buffers are in Elan memory, the asymptotic bandwidth is 280 MB/sec, a slight performance degradation from the 335 MB/sec of the unidirectional ping. With main memory to main memory communication the bandwidth drops to 80 MB/sec, a performance degradation caused by the PCI bus of the Alphaserver[6] , that is not able to efficiently interleave the bidirectional traffic. Given this substantial limitation, in the following experiments we will place the communication buffers in Elan memory, in order to

---

[6]Other PCI buses running at 66Mhz rather than at 33 Mhz, for example those based on the Serverworks HE chipsets, don't suffer from these limitations and can provide almost the same communication performance when the buffers are placed in main or in Elan memory [13].
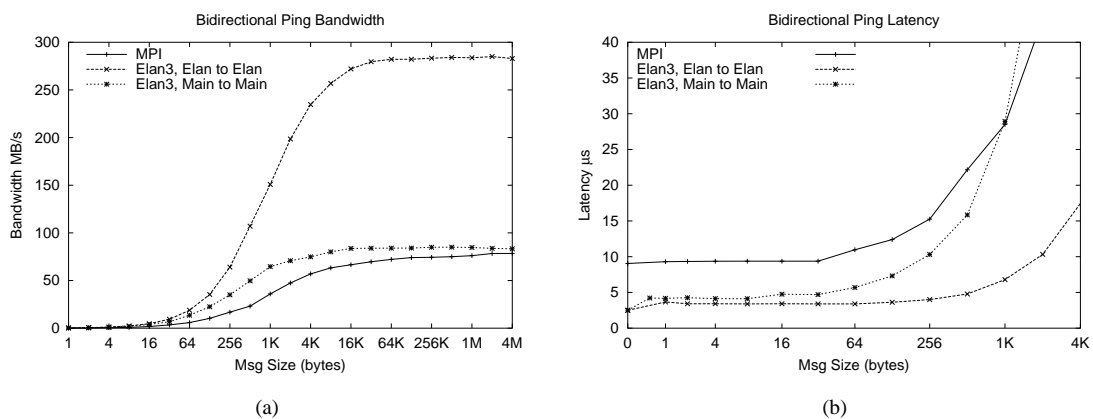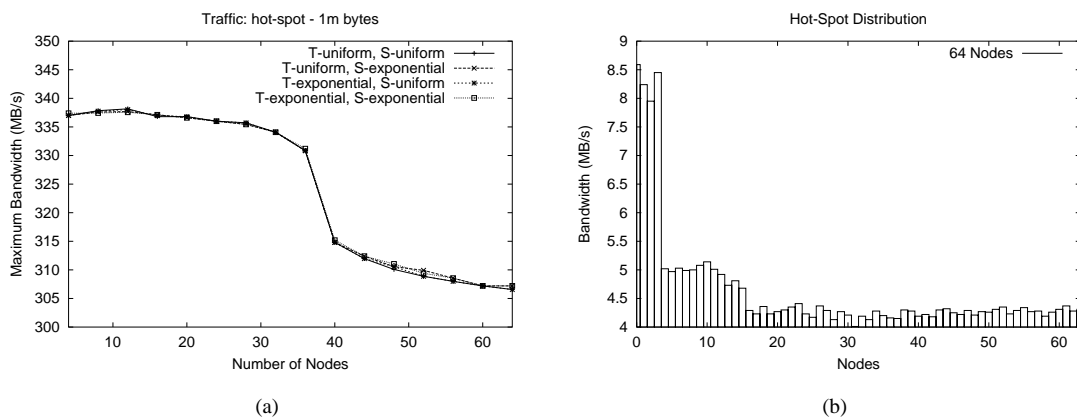
Figure 5. Bidirectional Ping

Figure 6. Write Hot-spot

inject messages into the network at full speed.

4.3 Hot-spot

In this experiment we attempt to write into the same memory location on node 0 from an increasing number of processors (one per SMP). This test provides information on the behavior of a single I/O node when serving multiple simultaneous requests. Previous results [13] have shown that read and write operations provide no significant differences. The aggregate bandwidth plots are depicted in Figure 6 a) for 1 MByte messages, using uniform and exponential time (T tag) and message size (S tag) distributions.

The curves are approximately flat up to 32 nodes, reaching 337 MB/s, while an 8% decrease is observed for 64 nodes. This is due to the extra contention experienced in the third level of switches (Figure 1 c)) and to the longer delays needed to release a circuit when there are more than 40 communication partners. Figure 6 b) shows the distribution of the delivered bandwidth per node on a 64-node configuration, and provides more insight into this problem. It can be seen that the nodes are distributed in three bandwidth groups: nodes from 0 to 3 get approximately 8 MB/s, nodes 4 to 15 get approximately 4.8 MB/s and nodes 16 to 63 get around 4.3 MB/s. This uneven distribution of bandwidth is due to the various areas of contention that a given packet can traverse. Considering that 0 is the hot node, packets sent from nodes 1 to 3 have a single contention point in the switch they are directly connected. Packets sent from nodes 4 to 15 have two potential contention points, one in the second level of switches and the other in the destination switch. Finally, packets sent from nodes 16 to 63 traverse three contention stages in the three levels of switches.

	Clustered I/O	Distributed I/O
Uniform Traffic	196 MB/s	234 MB/s
Fixed Traffic	320 MB/s	338 MB/s

Table 1. Multiple Hot-spots Maximum Accepted Load Summary
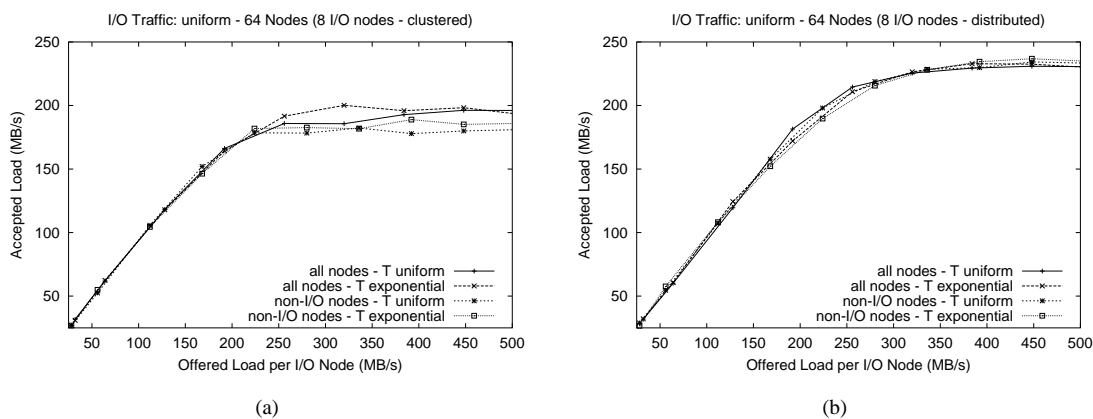

(a)   (b)

Figure 7. Multiple Hot-spots with Uniform Traffic

4.4 Multiple Hot-spots

This test is designed to analyze the behavior of the network when one parallel application is performing transactions over several I/O nodes, as described in Section 3.4.

The experiments are performed on a 64-node configuration with 8 hot-spots, as shown in Figure 2. The average message size is 1 MB (exponentially distributed), with inter-arrival times uniformly and exponentially distributed. The results show very little sensitivity to the fraction of read/write requests, so we will omit the related experiments. Figures 7 and 8 display the accepted load of the I/O nodes versus the offered load for uniform and fixed traffic, respectively. In each figure there is a graph for the clustered I/O mapping (sub figure a)) and a graph for the distributed I/O mapping (sub figure b)). Each graph displays curves for the two application mappings presented in Section 3.4 (all nodes and non-I/O nodes). The asymptotic bandwidths are summarized in Table 1 for the most significant performance dimensions, I/O traffic and I/O mapping.

The results show that a fixed destination pattern (Figure 8) always provides better performance than a random selection of destinations (Figure 7).

The I/O mapping has a significant effect on performance too. Better results are always obtained with distributed I/O (Figures 7 b) and 8 b)). This is due to the fact that the distribution of I/O nodes through the cluster evenly spreads the traffic across the network, while with the clustered mapping we generate a large hot-spot on one side of the network. It is worth noting that with distributed mapping each I/O node is connected to a distinct switch, while with clustered I/O four I/O nodes share the same switch. Thus, the adjacent allocation of I/O nodes worsens the contention in the network.

The application mapping has no significant effect when using distributed I/O (Figures 7 b) and 8 b)) and a small effect with clustered I/O (Figures 7 a) and 8 a)). In the latter case, the I/O nodes deliver slightly higher asymptotic bandwidth when all-nodes mapping is used (between 15 and 20 MB/s), either with fixed or uniform traffic.

4.5 Combined Traffic

With this benchmark we study how a parallel job that is executing I/O traffic can affect the communication performance of another parallel job that is running concurrently.

The I/O job generates uniform traffic, shown to produce high network contention (Section 4.4), with exponential distribution of the inter-arrival times and messages of 1 MB. We consider three I/O loads, with increasing intensity, which are expressed as fraction of the asymptotic load that can be injected into the network by a node (0.1, 0.3 and 0.5). The other parallel job uses uniform traffic and 256KB messages.
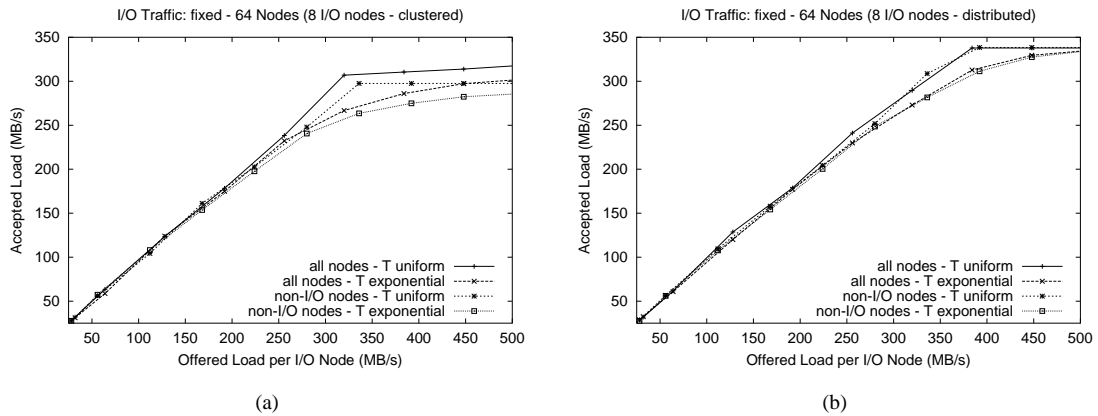
Figure 8. Multiple Hot-spots with Fixed Traffic

With clustered I/O, we distinguish two cases: the I/O job overlaps with the I/O nodes, and the symmetric case where the compute job overlaps with the I/O nodes (Section 3.5). In the graphs we use the label *1c*[7] to indicate the first case and *1i* to indicate the second case.

The graphs in Figure 9 show the accepted bandwidth of the compute job. Considering the type of traffic (uniform) and the message size, the optimal asymptotic bandwidth of the compute job is about 130 MB/sec. Any performance degradation indicates that the background I/O traffic interferes with the compute job. Figures 9 a), c) and e) consider the all nodes mapping, where the parallel jobs are also mapped onto the I/O nodes. In Figures 9 b), d) and f) we can see the results for the non-I/O nodes mapping. We can clearly see that :

1. When the jobs do not run on the I/O nodes (the application mapping is non-I/O nodes) there is no interference. This is a very powerful result that shows that any job mapping and I/O mapping will perform well, as long as the processes of both jobs do not run on the I/O nodes.

2. This basic result is extended to another case. When the compute job is not mapped onto the I/O nodes (clustered 1c, the I/O job overlaps the I/O nodes), there is no interference.

3. When a fraction of the compute job is mapped onto the I/O nodes (clustered 1i, the compute job overlaps the I/O nodes) there is a substantial performance degradation, up to 40%, with any I/O load.

4. With distributed mapping the performance is sensitive to the I/O load. The higher the background I/O load the lower the accepted bandwidth.

Figure 10 summarizes the analysis of the results and highlights the performance regions where the computational job is affected by the background I/O traffic.

5   Conclusions

In this paper we presented an extensive performance evaluation of several types of I/O traffic on a 64-node Alphaserver cluster interconnected in a fat-tree topology by the Quadrics network. This analysis can prove particularly useful for system and network designers and users of high-performance parallel clusters.

We first considered a single parallel job performing I/O and modeled the I/O traffic with a single hot-spot, representing a single I/O server, and multiple hot-spots, representing groups of I/O servers. The experimental results provided insight on several important open problems. We have shown that it is more efficient to distribute the I/O servers rather than cluster them in a single segment of the network, with a bandwidth increase of about 20%. Also, the performance is insensitive to both the fraction of I/O reads and writes and to the mapping of the parallel job, whose processes can be run on the I/O nodes without any noticeable performance degradation.
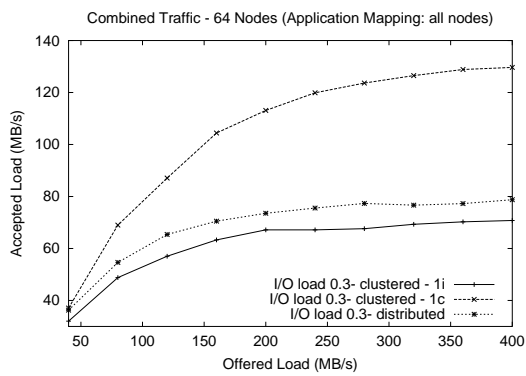
---

[7]A shortcut to indicate that the compute job is allocated on the first half of the machine and the I/O job on the second half, with the I/O nodes clustered in the last segment of the network.
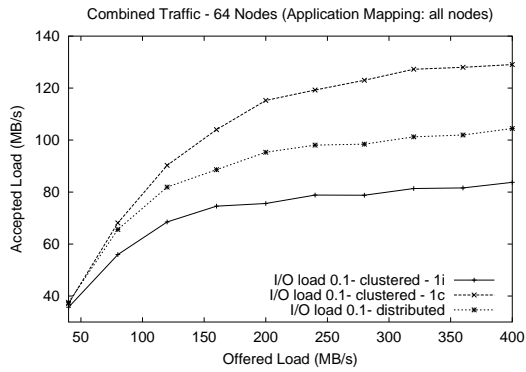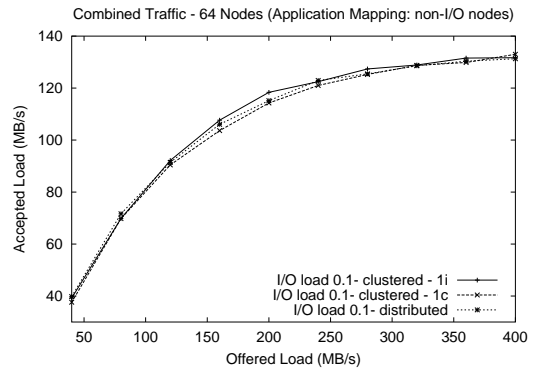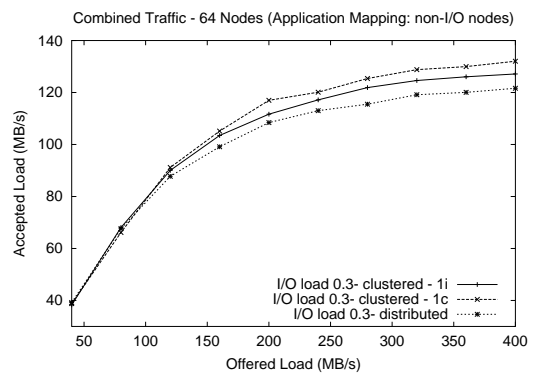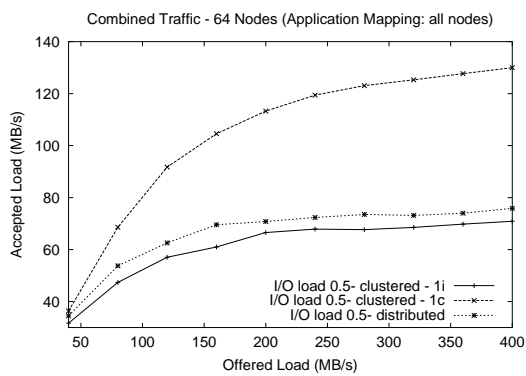
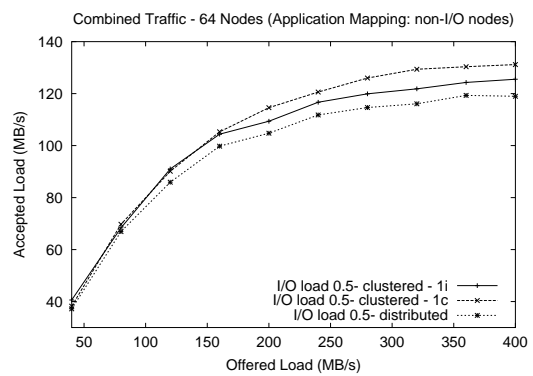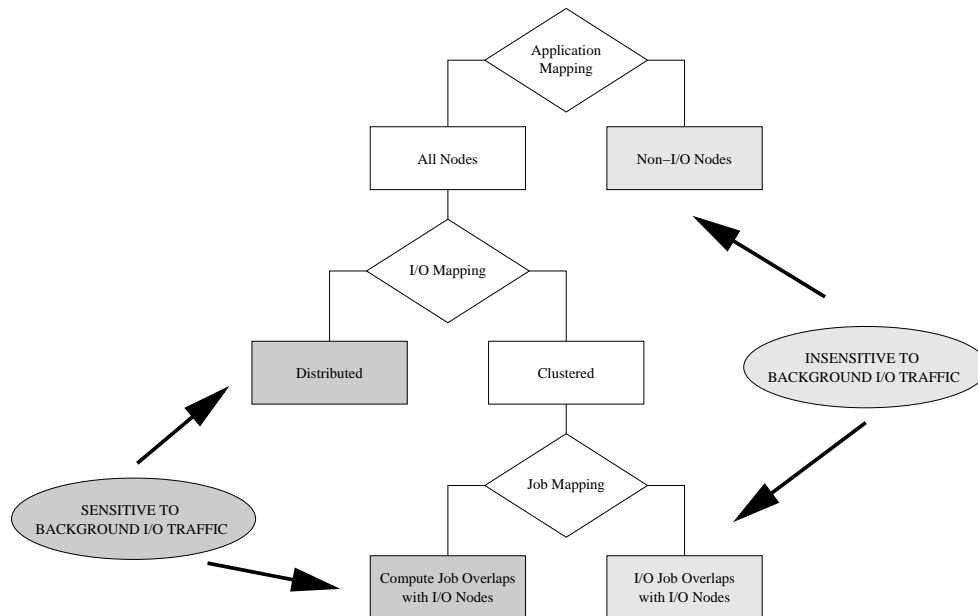Figure 9. Combined Traffic.

Figure 10. Analysis of the experimental results with combined traffic

We then analyzed how a job performing I/O can affect the communication performance of another job. Multiple jobs can be run concurrently without interference, as long as these jobs are not mapped on the I/O nodes. This is a powerful result, that gives a high degree of freedom when mapping multiple jobs. On the other hand, the I/O job can interfere with a compute job when some processes of the compute job are mapped on the I/O nodes.

Acknowledgments

References

[1] *InfiniBand Specification 1.0a*. InfiniBand Trade Association, June 2001.

[2] Rajesh Bordawekar, Steven Landherr, Don Capps, and Mark Davis. Experimental Evaluation of the Hewlett-Packard Exemplar File System. *ACM SIGMETRICS Performance Evaluation Review*, 25(3):21–28, December 1997.

[3] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur. PVFS: A Parallel File System for Linux Clusters. In *4th Annual Linux Showcase and Conference*, pages 317–327, October 2000.

[4] Yong Cho, Marianne Winslett, Szu wen Kuo, Ying Chen, Jonghyun Lee, and Krishna Motukuri. Parallel I/O on Networks of Workstations: Performance Improvement by Careful Placement of I/O Servers. In *HiPer '98, High Performance Computing on Hewlett-Packard Systems*, Zurich, Switzerland, October 1998.

[5] Peter F. Corbett, Dror G. Feitelson, Jean-Pierre Prost, George S. Almais, Sandra Johnson Baylor, Anthony S. Bolmarcich, Yarsun Hsu, Julian Satran, Marc Snir, Robert Colao, Brian Herr, Joseph Kavaky, Thomas R. Morgan, and Anthony Zlotek. Parallel File Systems for the IBM SP Computers. *IBM Systems Journal*, 34(2):222–248, January 1995.

[6] William J. Dally and Charles L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[7] Adolfy Hoisie Salvador Coll Fabrizio Petrini, Wu-chun Feng and Eitan Frachtenberg. Quadrics Network (QsNet): High-Performance Clustering Technology. In *Hot Interconnects 9*, Stanford University, Palo Alto, CA, August 2001.

[8] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. *MPI - The Complete Reference*, volume 2, The MPI Extensions. The MIT Press, 1998.

[9] Intel Corporation. *Intel Scalable Systems Division. Paragon System User's Guide*, may 1995.

[10] Darren Kerbyson, Hank Alme, Adolfy Hoisie, Fabrizio Petrini, Harvey Wasserman, and Mike Gittings. Predictive Performance and Scalability Modeling of a Large-Scale Application. In *Supercomputing 2001*, Denver, CO, November 2001.

[11] Charles E. Leiserson. Fat-Trees: Universal Networks for Hardware Efficient Supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.

[12] Fabrizio Petrini, Salvador Coll, Eitan Frachtenberg, and Adolfy Hoisie. Hardware- and Software-Based Collective Communication on the Quadrics Network'. In *IEEE International Symposium on Network Computing and Applications 2001 (NCA 2001)*, Boston, MA, October 2001.

[13] Fabrizio Petrini, Adolfy Hoisie, Wu chun Feng, and Richard Graham. Performance Evaluation of the Quadrics Interconnection Network. In *Workshop on Communication Architecture for Clusters (CAC '01)*, San Francisco, CA, April 2001.

[14] Fabrizio Petrini and Marco Vanneschi. k-ary n-trees: High Performance Networks for Massively Parallel Architectures. In *Proceedings of the 11th International Parallel Processing Symposium, IPPS'97*, pages 87–93, Geneva, Switzerland, April 1997.

[15] Fabrizio Petrini and Marco Vanneschi. Performance Analysis of Wormhole Routed k-ary n-trees. *International Journal on Foundations of Computer Science*, 9(2):157–177, June 1998.

[16] G. F. Pfister and V. A. Norton. Hot-spot Contention and Combining in Multistage Interconnection Networks. *IEEE Transactions on Computers*, C-34(10):943–948, October 1985.

[17] Kenneth W. Preslan, Andrew P. Barry, Jonathan E. Brassow, Grant M. Erickson, Erling Nygaard, Christopher J. Sabol, Steven R. Soltis, David C. Teigland, and Matthew T. O'Keefe. A 64-bit. Shared Disk File System for Linux. In *Seventh NASA Goddard Conference on Mass Storage Systems*. IEEE Computer Society Press, March 1999.

[18] Quadrics Supercomputers World Ltd. *Elan Reference Manual*, January 1999.

[19] Quadrics Supercomputers World Ltd. *Elite Reference Manual*, November 1999.

[20] SGI. *XFS: a next generation journalled 64-bit filesystem with guaranteed rate I/O*. http://www.sgi.com/Technology/xfs-whitepaper.html.

[21] Hal Stern. *Managing NFS and NIS*. O'Reilly & Associates, Inc., 1991.