Peter Strazdins, Richard Alexander, and David Barr. *Performance Enhancement of SMP Clusters with Multiple Network Interfaces Using Virtualization*.
G. Min et al. (Eds.): ISPA 2006 Ws, LNCS 4331, pp. 452–463, 2006.

This work will be published under Springer-Verlag *Lecture Notes in Computer Science*,
`http://www.springerlink.com`

Performance Enhancement of SMP Clusters with Multiple Network Interfaces using Virtualization

Peter Strazdins[1], Richard Alexander[2], and David Barr[2]

[1] Department of Computer Science, Australian National University
Peter.Strazdins@cs.anu.edu.au
[2] Alexander Technology, Canberra,
http://www.alexandertechnology.com,
{richard,david}@alexandertechnology.com

Abstract. Clusters of small-scale SMP/CMP nodes are becoming increasingly popular due to their cost-effectiveness. As these nodes are typically capable of supporting a number of network interfaces similar to the number of CPUs, the issue arises how to optimally configure the cluster for optimum communication performance. This paper evaluates a number of configurations on a 4-CPU Opteron cluster with multiple Gigabit Ethernet interfaces. Techniques include channel bonding and using independent communication pathways. With the latter, the use of virtualization via the Xen Virtual Machine Monitor offers the best potential to parallelize all stages of message transmission, for the case when multiple CPUs on a node are communicating simultaneously. Network-level microbenchmarks indicate the best performance is achieved with a configuration where guest virtual machines running on each CPU communicate directly with a dedicated interface, bypassing the virtual machine monitor. Channel bonding also proved to be more effective over multiple communication streams than over single.

1 Introduction

Cluster computers, assembled from COTS commodity-off-the-shelf compute nodes and communication networks, have proved a highly cost-effective solution to high performance computing demands, and have gained dominance in this market. COTS technology has provided high increases in computational speed for a given cost, but in terms of communication networks, the definition of COTS is not only less clear, but their performance increase has not matched that of the compute nodes. While (Gigabit) Ethernet continues to be the most widely used (and most strongly fits the COTS criterion) communication network, there are more specialized networks available, such as InfiniBand, Myrinet and Quadrics.

While low-end SMP nodes have long been seen as highly cost-effective in the cluster context [1–3], the recent advent of Chip Multiprocessing (CMP) promises an even higher price-performance advantage. CMP adds a new dimension to the question of the optimal number of CPUs for a cluster computer node. The key issue is that the communication and computational performance must be balanced for a cluster configuration to be cost-effective.

However, many of the COTS processor systems which may be selected for a cluster compute node come with motherboards supporting multiple I/O connections. Typically on these motherboards, the main system bus is connected to a number of PCI buses, each of which may have a number of slots where I/O devices can be connected. For example, the IWILL DK8-HTX motherboard for Opteron systems has AMD-8111 I/O Hub and AMD-8131 PCI-X Tunnel chips, between them having three PCI-X/PCI buses and 14 device slots, as well as two in-built Gigabit Ethernet controllers [4]. Since network interfaces are themselves I/O devices, this permits a considerable number of interfaces to be connected[3], which can potentially provide an aggregate communication bandwidth to match the nodes' aggregate compute performance for a moderate number of CPUs. As the network's cost (interface cards and switches) is typically a small fraction of an Ethernet-based cluster's overall cost [1], multiple network interfaces may prove to be similarly cost effective as multiple CPUs. This is particularly the case when the ports of the switch are under-utilized, in which case the extra cost is only in the cards, which for Gigabit Ethernet is typically under a few hundred dollars each.

In recent years, there has been a resurgence of interest in *virtual machines* (virtualized operating systems), largely due to the increased encapsulation that this offers, which in turn offers advantages in flexibility, security, performance isolation and migration [5, 6]. There are various techniques which may accomplish this, but one offering both potentially high performance and high functionality is known as *paravirtualization*. Xen [5] is an x86-based virtual machine monitor for Linux which uses this technique. The para-virtualization approach of Xen offers an easy way of dedicating network interfaces to instances of virtual machines (which in turn may be running simultaneously on multiple CPUs).

This paper is concerned with an increasingly important issue in cluster design of how to determine the optimal number of CPUs and network interfaces per node, and of how to configure the interfaces. To this end, the paper evaluates various multiple network interface configurations on an SMP cluster with at least as many CPUs. In terms of configurations, we explore two broad possibilities for multiple CPU nodes: *channel bonding*, where all interfaces may be used to send parts of a message (individual packets, in the case of Ethernet [7]); and setting up independent network interfaces for a particular source (or destination) CPU. Our emphasis is on Gigabit Ethernet interfaces, due to their relatively low cost and wide deployment. A key issue in this context is the degree of parallelization possible over the stages of message transmission: the TCP/IP stack, the access of the network interfaces (either network interface cards (NICs) or chips), and the transmission across the communication channel. PCI bus configurations can also play an important role in this process. For independent network interface configurations, dedicating network interfaces to virtual machine instances offers the potential of parallelization over all stages of message transmission. Thus, this paper is will also explore the potential benefits, and overheads, of virtualization in these configurations.

This paper is organized as follows. Section 2 discusses related work, and defines the new contributions made in this paper. Background information on Xen and TCP/IP

[3] Although in practice, bus bandwidth limitations and the number of interrupt requests available would limit this number.

stack processing is given in Section 3. A variety of multiple network interface configurations that we will study is described in Section 4, with the experimental setup described in Section 5. Performance results are given in Section 6 and conclusions are given in Section 7.

2 Related Work

There are a number of performance evaluations of cluster networks with SMP nodes in the literature (e.g. see [2, 8] and the references within). These typically evaluate the effect of connecting cluster nodes with different networks (interface card and switch combinations), but use configurations with a single network interface for communication.

Gigabit Ethernet-based networks, due to their popularity, have also been evaluated. [7] examines the effects of channel bonding of a dual Xeon connected with dual Intel/Pro 1000 ports; it concluded that the channel bonding provided by the Linux kernel was mostly ineffective, and even degraded performance for medium-sized messages. However, it concluded that the related technique of *striping* the data at the socket level (which permits more independent TCP/IP stack processing for each interface) could almost double the bandwidth.

Network I/O performance has recently been recognized as an important issue for Xen [9]. Here, a multiple TCP stream configuration showed that the Xen 'driver domain' (see Section 3) achieved 69% and 100% of the native Linux's receive and send performance, respectively; whereas a normal (guest) VM under Xen achieved 33% and 20% respectively. Subsequent optimizations improved the driver domain's receive performance to 90%, and the VM's send performance to 90% [10]. The configuration used here is the most similar to ours so far, in that the experiment aggregated the performance of 4 server processes, each connected to an independent NIC (c.f. the `*.indep.4p` configurations of Section 4). However, the server was not an SMP, and the clients were on 4 separate machines; thus, their configurations emphasise the CPU overheads of Xen more than ours.

There has been recent interest in the use of virtualization for cluster computing. Key issues include reducing the performance (particularly for message passing) and management overheads, with preliminary solutions being proposed and evaluated [6]. The solution for reducing messaging passing overheads is called *virtual machine monitor bypass* (VMM-bypass), and is elaborated in [11]. This solution can be applied to networks with OS bypass capabilities (also known as *user-level communication*), in this case InfiniBand, which can similarly be used to bypass the virtual machine monitor. The results show that the performance of communication under bypass of the Xen monitor approaches that of the original InfiniBand driver. However, there is not a clear evaluation of how large the overheads were originally under Xen without bypass.

This paper's contributions are that it makes a comparison of various multiple network interface configurations for clusters with multiple CPU nodes. Techniques used include channel bonding and VMM-bypass; however the latter is used to set up independent communication channels, and is a more generic approach as it does not require

OS bypass capabilities of the network. In the comparison of the multiple configurations, we also evaluate the overhead of Xen at the MPI level.

3 Background

This section gives background information which is relevant to the experiments on various the GigE network interface configurations described subsequently.

Various references [5, 12, 11] describe the approach of Xen to virtualization, which the reader is referred to. Xen requires one special guest VM, called *domain0*, to be present; this is used to manage a number of guest VMs. These guest VMs can communicate to each other using shared pages; communication to external VMs occurs through *a virtual interface*. Data is transferred via pseudo-device drivers to domain0; by default, only this domain has access the native device interfaces. Apart from the processing of interrupts, which are fielded first by the VMM, device access from domain0 proceeds very similarly as it would under the corresponding Linux kernel that XenoLinux is based on. For this reason, domain0 is also referred to as the 'driver domain' [10].

The Linux kernel 2.6 has sophisticated TCP/IP stack processing on multiple CPU systems. Due to its widespread importance, studies have recently emerged analysing the parallelization strategies used in Linux [13, 14]. Two kinds of locks are required for TCP/IP stack processing: locks related to connection, and locks associated with particular sockets, with the latter typically requiring more frequent access.

Two broad parallelization strategies exist: *connection-parallel* and *message-parallel*, with the former being regarded as superior [13, 14]. The message-parallel strategy parallelizes the processing (of different segments) of a single transmission; it can be employed when channel bonding is in use, but typically requires large message sizes to become effective. The connection-parallel strategy allows messages using different sockets to proceed in parallel; this eliminates contention on the per-socket locks [13], and thus explains why socket striping achieves better performance than channel bonding (as reported in [7]).

4 Multiple Network Interface Configurations

Figure 1 shows the configurations for nodes with 2 network interfaces (and 2 CPUs). The acronym NIC should be regarded here as denoting any kind of network interface, whether implemented on a card or on a chip (note also that some cards have dual interfaces). In each case, there are MPI process pairs 0 and 2, and 1 and 3; it can be assumed that communication only occurs between pairs. The diagram indicates communication paths between these processes, rather than physical connectivity. In fact, both connection paths (denoted Nic1 and Nic2) go through the same switch, and in all cases it is possible for any process to communicate with any other.

Configuration `driver.bond.2p` runs MPI processes on the driver domain (domain0), using channel bonding to combine the aggregate performance of the two connecting interfaces. Configuration `driver.indep.2p` is similar, except traffic between the pairs occurs independently on separate NICs; this is achieved by each node

being given two IP addresses and binding each to two separate Ethernet interfaces (e.g. `eth1` and `eth2`). The `route add` command is used to route traffic to each of the IP addresses of the other node through one of these interfaces. MPI processes are then configured using a list of the four IP addresses. This ensures that different sockets are used for each stream, thus providing a simple way of ensuring that a connection-parallel strategy is employed.

Configuration `guest.bond.2p` requires two Xen guest domains (1 and 2) to be configured on each node, with the MPI processes being assigned to each of the four VMs (with each assigned a different IP address). Communication occurs through domain0, which uses bonding of the two NICs. Configuration `guest-byp.indep.2p` is similar, except it uses VMM-bypass: i.e. the NICs are unbound from domain0 and each independently bound to one of VMs. As Xen binds a VM to a CPU (see Section 4.1), this results in processes and NICs being automatically bound to a specific CPU.

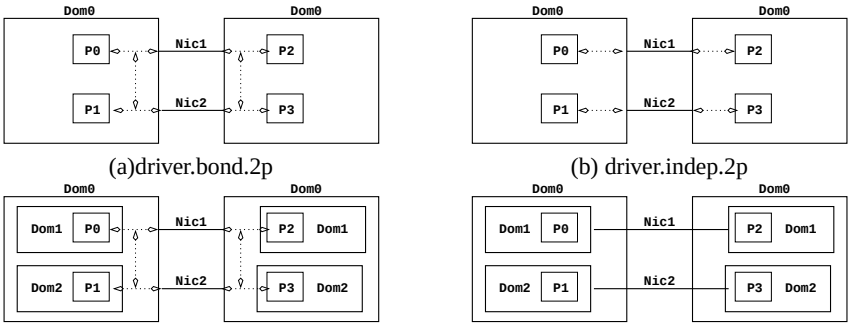


Fig. 1. 2-way NIC configurations

For a baseline comparison, there are also 2 process (1 pair) versions of these configurations, denoted similarly but with the suffix '`.1p`'. There are similarly 4 pair versions of the above configurations, which use 4 NICs (and use 4 CPUs, with 4 IP addresses per node), denoted with the suffix '`.4p`'.

In terms of parallelization of the TCP(/IP) stack, the `guest-byp.indep.*` configurations offers fully independent processing for multiple network interfaces. The host-indep.* configurations offer socket-level parallelism, whereas the *.bond.* configurations offer connection-level parallelism.

The characteristics of intra-node communication are also of interest (this would correspond to a situation as on Figure 1, except `P0` and `P2` are on one node, and `P1` and `P3` are on the other). For the `driver.*` configurations, these will occur via a shared memory transport. For the `guest*.*` configurations, these occur via *virtual interfaces*, implemented in turn by *event channels*, which can exchange data by shared pages [11]. If this is the case, while there would be some overhead of invoking the Xen

VMM (to service requests in the event channels), there need be little or no copying overhead for the data.

4.1 Modifying Xen for domain0 Bypass

Normally, guest domains under Xen perform I/O via a *virtual interface* to domain0; domain0 then accesses these hardware devices directly [12]. This is done by setting the `vif` variable in the guest domain's configuration file.

If this is omitted, no virtual interfaces are set up; however, an actual (PCI-connected) network interface can be set up to perform I/O instead [12]. This can be specified in the guest domain's configuration file by setting the `pci` variable to the desired bus and slot number, e.g. to connect to the card on slot 4 of PCI bus 3, the setting is `pci = [ '03,04,0' ]`.

The binding of a guest domain to a single CPU is similarly specified in its configuration file. It remains to ensure that before the guest domains are brought up, these slots are unbound from domain0, and, for maximum efficiency, the interrupt requests arising from that slot are directed to the same CPU. In Linux, this can be done by creating a file `/proc/irq/`i`/smp_affinity` which contains the CPU's number, where i in the interrupt request number of that slot.

Note that in this context, the VMs form the nodes of a (virtual) cluster which will be used to run parallel jobs and so must 'trust' each other; thus, there is less of a security issue here in bypassing Xen's driver domain.

5 Experimental Setup

We use a 2 node cluster for our experimentation. Each node consists of dual SMP dual-core 2.2 GHz AMD Opteron processors with a 2-way 64 KB level 1 data cache and an 8-way 512 unified L2 cache, and 4 GB of RAM. The nodes have an IWILL DK8-HTX_815 motherboard, with 800 MB/s HyperTransport links. The motherboard has in-built dual Intel 82541GI/PI GigE controllers. External NICs can be connected to two slots connected to the same 64-bit 33/66/100 MHz PCI-X bus, and to one slot connected to a third 32-bit 66 MHz PCI 2.2 bus. For external NICs with dual Ethernet ports, this permits up to 8 Gigabit Ethernet interfaces on this motherboard.

One of the in-built GigE chips is configured to Ethernet interfaces `eth1` (the other is needed by the driver domain as a control interface). A Pro/1000 MT NIC with an Intel 82541PI chip is configured to `eth4`. The nodes also have a Pro/1000 MT NIC (Intel 82546GB chips) with dual interfaces; these are configured to `eth2` and `eth3`. Inspection of the Linux device driver code indicates that the 82546 chip supports *segmentation bypass*, i.e. offload of some of the IP stack, but that there is no offload for the 82541 chip. Note that the same device driver is used for all interfaces. The `lspci` command indicates `eth1`, `eth2` and `eth3` are on PCI bus 3, which is running at 66 MHz in 32-bit mode (total bandwidth of 240 MB/s), and that `eth4` is on PCI bus 1, also running at 66 MHz in 32-bit mode.

The networks are configured with am Ethernet Message Transfer Unit of 4148 bytes - sufficient to hold a 4KB payload under TCP/IP. This value was found to be optimal under elementary network bandwidth tests.

The system software is based on the Linux Dapper Drake 6.0.6 distribution, with a XenoLimux 3.2.2 kernel. This is based on the Linux kernel 2.6.16 SMP, which both the driver and guest domains are based on. gcc 4.0.3 comes with this distribution. The channel bonding driver, when used, is that which comes with the Linux kernel.

5.1 Benchmark Programs

Our benchmark programs use the MPICH-2 MPI implementation under the MVAPICH-2 package from the Ohio State University [15]. MPI is configured to only use interfaces `eth1` to `eth4`.

To test raw communication performance, we use the latency and bandwidth (uni- and bi-directional) benchmarks program, also available under the MVAPICH-2 package [15, 8]. The latency tests give the averaged timings for $r = 100$ ping-pong tests[4]. The uni-directional bandwidth test is for $g = 64$ one-way messages followed by an acknowledgement, repeated $r = 20$ times. The bi-directional bandwidth tests are similar, except each nodes posts g receives and then sends g messages. All tests have a warm-up period of $r/10$ un-timed messages (of the same length as those to be measured).

The latency test is useful as it represents the communication performance of where a node on the 'critical path' of a parallel computation is waiting on a message. The uni-directional bandwidth test is aimed to demonstrate the maximum one-way bandwidth performance. It models pipelined communication, such as is used in applications such as parallel Linpack. The bi-directional tests can show saturation effects (in the PCI bus and/or network interfaces); it also models important communication patterns such as all-to-all exchange.

The benchmarks use `MPI_Wtime()` to measure time, which in this case, is based on `gettimeofday()`. This should return the actual wall time irrespective of whether running on real or virtualized hosts.

These benchmarks normally run as 2 MPI processes. For our experiments, they were modified to run as pairs of MPI processes; in this case 1, 2, or 4 pairs being of interest. Also, calls to `sched_setaffinity()` were used to bind processes to the CPU corresponding to their process number. Timings for each pair are recorded. Our methodology involves performing 10 timings for each data point on an otherwise quiescent system; while the average is of most interest, variations may indicate 'stress' on the kernel and/or an asymmetry in the loads over each CPU from message processing.

5.2 System Integrity Issues

The use of 4 network interfaces in the 4 CPU nodes initially created problems. It caused communication-intensive applications to hang or crash. The problem was traced to the handling of interrupt requests, due to the limited number available on the DK8-HTX motherboard. Some local disk I/O interrupts, as they shared the same interrupt number as one of the network devices were lost.

The solution was to configure the operating system (domain0) to implement its filesystem on a network bootable RAM disk rather than the local disk.

[4] r is increased by a factor of 10 for small messages.

This problem does however indicate a practical limitation to the number of network interfaces that can be used simultaneously on a given motherboard.

6 Results

In Section 6.1, we report the performance of the single pair configurations. Native Linux performance is also included and the bandwidths of the separate interfaces are also measured. From this, we can evaluate the overhead of Xen and gain a baseline to understand the performance of multiple interfaces, which is presented in Section 6.2.

6.1 Baseline Performance

For single pair performance (interface `eth1`), the unidirectional bandwidth approaches 110 MB/s, close to the theoretical maximum of Gigabit Ethernet (125 MB/s), for all but the `guest.bond.1p` configuration. The bi-directional case is similar, except the bandwidth approaches 135 MB/s. In both cases, the `*.indep` configurations perform best, being virtually indistinguishable from each other. Similar to [7], channel bonding in the driver domain gives lower performance, especially in the 1KB – 256 KB region, but in the unidirectional case for messages over 1 MB it has slightly faster performance. The performance of the `guest.bond` configuration, which performs communication via Xen's virtual interfaces to domain0 (which in turn is configured to use channel bonding) approaches 50MB/s in both cases – much lower than the others; this trend we will see maintained. The bandwidth across `eth2` and `eth3` was about 15% faster in the bi-directional case, and `eth4` was slower with a maximum bandwidth of 90 MB/s.

For the latency tests, the results are similar to the unidirectional bandwidth case, with the bandwidth increasing sharply at 1 KB. However, `driver.bond` suffers a greater (30-50%) performance loss over the 32 – 256 KB range. Table 1 summarizes performance at the endpoints.

	`*.indep`	`driver.bond`	`guest.bond`
time at 1 B (μs)	87	91	106
B/W at 4 MB (MB/s)	107	105	51

Table 1. Inter-node Latency Test Performance summary for 1-pair configurations

The above experiments were also run under native Linux (based on the same kernel and distribution as for XenoLinux) with independent channels; we call this the `native.indep.1p` configuration. The results were identical to `driver.indep.1p`, except that the latency test yielded a marginally higher bandwidth of 109 MB/s (at 4 MB); this indicates the messaging performance under domain0 is essentially identical to native Linux.

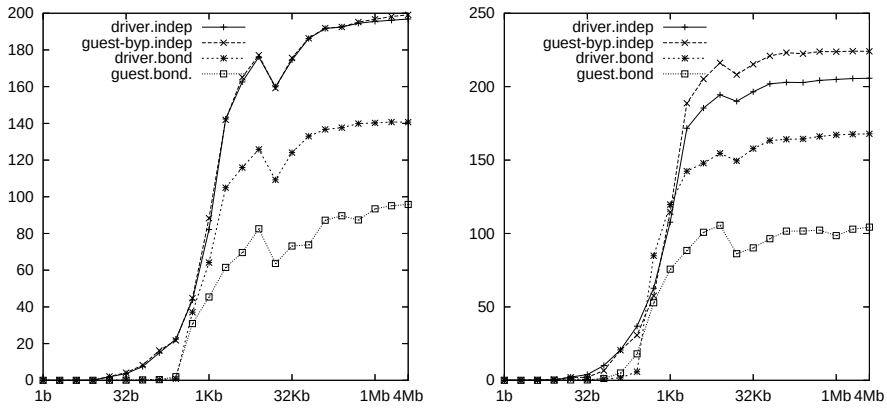
Intra-node performance was measured similarly, by running the MPI processes on the same physical host; the results a summarized in Table 2. The `driver.*` configurations use a shared memory transport; the `guest*.*` configurations use virtual

interfaces; by comparison with Table 1, this has a very similar bandwidth to that of inter-node communication. The uni- and bi-directional bandwidths for large messages were the same as for the latency tests.

	`driver.*`	`guest*.*`
time at 1 B (μs)	18	530
B/W (MB/s) at 4 MB	30	50

Table 2. Intra-node latency test performance summary for 1-pair configurations

6.2 Multiple Interface Performance



(a) uni-directional (b) bi-directional

Fig. 2. Bandwidth (MB/s) versus message size for 2-pair configurations

Figure 2 gives the bandwidths for 2 pair performance (using interfaces `eth1` and `eth4`). Figure 3 gives the performance for 4 pairs; configuration `guest.bond.4p` is omitted here, due to it being insufficiently stable to complete the tests.

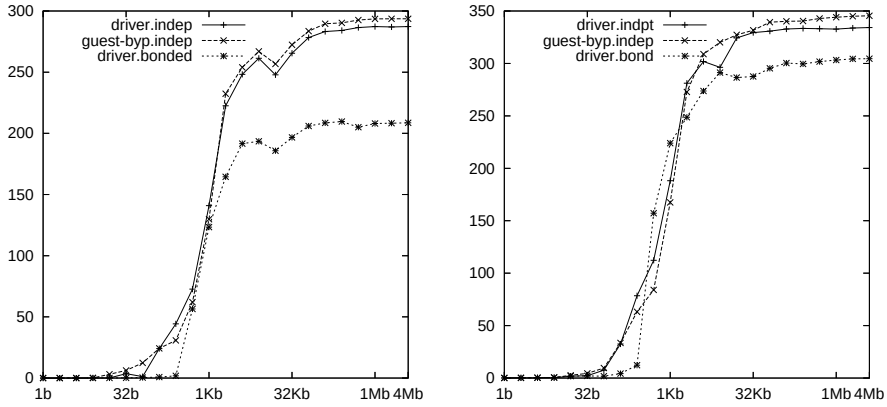
For the latency tests (not graphed), the results are similar to the 1-pair tests, with again `driver.bond` suffering a 30-50% performance loss over the 32 – 256 KB range. Table 3 summarizes performance for large messages.

Overall, it can be seen that the `*.indep` configurations consistently give the best performance, with the `guest-byp.*` configuration performing the same on some tests, and $\approx 10\%$ better on others. In the 4-pair case, it achieves 300 MB/s and 345 MB/s for uni- and bi-directional bandwidth, respectively. This is comparable with the

	driver.indep	guest-byp.indep	driver.bond	guest.bond
2-pairs	172	200	144	86
4-pairs	296	296	256	–

Table 3. Latency test bandwidth at 4 MB for 2- and 4-pair configurations

maximum expected uni-directional bandwidth, which is 354 MB/s; this is the sum of the bandwidth on PCI bus 3 (264 MB/s) plus the bandwidth over `eth4` (90 MB/s – see Section 6.1).



(a) uni-directional (b) bi-directional

Fig. 3. Bandwidth (MB/s) versus message size for 4-pair configurations

Comparing the results in Section 6.1 and Figures 2–3, we see a clear (although sub-linear) increase in bandwidth as network interfaces (and communication streams) are added. In particular, `driver.bond` improves its performance as communication intensity increases, and also demonstrates benefit from multiple communication streams. However, from extrapolating the trends, it seems likely that there would be diminishing returns from adding more interfaces, at least on the motherboard used.

It should be noted that there was a significant variation in the measurements for the `driver.indep.4p` tests (the averaged results are over 10 measurements). The `driver.bond` configurations experienced some variations, but these were significantly reduced once CPU affinity was imposed. CPU affinity did not however seem to have a large impact on average performance. `guest-byp.indep` showed very small variability in all tests; it can be noted that affinity is enforced in the `guest*.*` configurations.

The `native.indep.2p` results were indistinguishable from that of the `driver.indep.2p`. However, the `native.indep.4p` results showed con-

flicting differences over `driver.native.4p`: the latency test showed a bandwidth of only 256 MB/s at 4MB, but the uni-directional bandwidth was much higher, peaking at 335 MB/s. Bi-directional bandwidth shows great variability in the 4–64 KB range, peaking at 382 MB/s at 8 KB, but decreased to 300 MB/s after 256 KB. In this situation, while communication performance is different in domain0 over native Linux, neither shows conclusively better performance overall.

7 Conclusions and Future Work

Our preliminary experiments on a 4 CPU Opteron-based Gigabit Ethernet clusters indicate that worthwhile improvements in communication bandwidth can be achieved by using multiple network interfaces. With one communication stream per process pair, configuring streams to run independently across separate interfaces generally yielded significantly better performance than did channel bonding. However, relatively better gains for channel bonding were observed as the number of streams and communication intensity increased. The best performance came from configurations using Xen virtualized hosts with VMM-bypass for network I/O. This is because it has the greatest potential of parallelism in TCP/IP stack processing, as well as providing natural affinity between CPU and network interface interrupt processing. However, in the setup used, its advantage over independent streams in Xen domain0 was not decisive.

While it is easy to enable VMM-bypass for network-based communication in Xen to optimize inter-node communication, intra-node communication between Xen guests on the same node is currently an order of magnitude slower than the native shared memory transport. This would counteract the advantages of VMM-bypass for configuring an SMP cluster with a Xen guest on each CPU.

Xen communication bandwidth, going through the VMM, is still generally within a factor of two of the best configuration possible. Communication performance of the Xen driver domain (domain0) closely matched that of native Linux for single network interfaces and 1-pair configurations, although for 4-pair configurations, there was some variability but no decisive overall difference.

Future work includes optimizing communication performance between Xen guests on the same node; this could be implemented as a shortcut to a shared-memory transport in the virtual interface implementation. Once this is done, application-level performance could be meaningfully evaluated over the configurations studied here. Other directions for future work include evaluating these effects on nodes of different motherboards; particularly interesting will be the 8 CPU case.

It is foreseeable that virtualization, with a combination of VMM-bypass and optimization, may actually offer performance advantages in SMP clusters. As well as permitting some advantages in average performance, configurations of one virtual machine per CPU show low variability in performance, due to the increased encapsulation afforded by para-virtualization.

Acknowledgements

The authors thank Alistair Rendell for helpful suggestions in the experimentation and the preparation of the manuscript. We also thank Tony Breeds for setting up the software distribution and some of the result-generating infrastructure used in this work, and thank Brendan Howe for technical support.

References

1. Aberdeen, D., Baxter, J., Edwards, R.: A 98c/MFLOP Ultra-Large Scale Neural Network Training on a PIII Cluster. In: Proceedings of Supercomputing 2000. (2000)
2. Capello, F., Richard, O., Etiemble, D.: Understanding performance of SMP clusters running MPI programs. Future Generation Computer Systems **17** (2001) 711–720
3. Pukayastha, A., Guiang, C.S., Schulz, K., Minyard, T., Milfeld, K., Barth, W., Hurley, P., Boisseau, J.R.: Performance Characteristics of Dual-processor HPC Cluster Nodes based on 64-bit Commodity Processors. In: Proceedings of the Linux Clusters Institute (LCI) International Conference: the HPC Revolution. (2004)
4. Advanced Microelectronic Devices: AMD Microprocessor Solutions. (http://www.amd.com/us-en/Processors)
5. Barham, P., Dragovic, B., Fraser, K., Harris, S.H.T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. In: Proceedings of SOSP 03: the Nineteenth ACM Symposium on Operating Systems Principles, Ney York, ACM (2003) 164–177
6. Huang, W., Liu, J., Abali, B., Panda, D.: A Case fopr High Performance Computing with Virtual Machines. In: Proceedings of ICS06: International Conference of Supercomputing, Cairns (2006)
7. Turner, D., Oline, A., Chen, X., Benjegerdes, T.: Integrating New Capabilities into NetPIPE. In: 10th European PVM/MPI User's Group Meeting, Venice, Springer (2003) 37–44
8. Liu, J., Chandrasekaran, B., Wu, J., Jiang, W., Kini, S., Yu, W., Buntinas, D., Wyckoff, P., Panda, D.: Performance Comparison of MPI Implementations over Infiniband, Myrinet and Quadrics. In: Proceedings of the SuperComputing 2003 Conference, Phoenix (2003)
9. Menon, A., Jose Renato Santos, a.Y.T., Janakiraman, G., Zwaenepoel, W.: Diagnosing Performance Overheads in the Xen Virtual Machine Environment. In: First ACM/USENIX Conference on Virtual Execution Environments (VEE'05). (2005) 13–25
10. Menon, A., Cox, A.L., Zwaenepoel, W.: Optimizing Network Virtualization in Xen. In: Proceedings of the 2006 USENIX Annual Technical Conference, Boston (2006) 15–28
11. Liu, J., Huang, W., Abali, B., Panda, D.: High Performance VMM-Bypass I/O in Virtual Machines. In: Proceedings of the 2006 USENIX Annual Technical Conference, Boston (2006)
12. University of Cambridge Computing Laboratory: The Xen virtual machine monitor. (http://www.cl.cam.ac.uk/Research/SRG/netos/xen)
13. Willmann, P., Rixner, S., Cox, A.L.: An Evaluation of Network Stack Parallelization Strategies in Modern Operating Systems. Technical Report TR06-872, Rice University Computer Science (2006)
14. Bhattacharya, S.P., Apte, V.: A Measurement Study of the Linux TCP/IP Stack Performance and Scalability on SMP systems. In: Proceedings of the 1st International Conference on COMmunication Systems softWAre and middlewaRE (COMSWARE), New Delhi (2006)
15. Nowlabs, Ohio State University: MVAPICH2 Toolset. (http://nowlab.cse.ohio-state.edu/projects/mpi-iba/)