

Performance Enhancement of SMP Clusters with Multiple Network Interfaces using Virtualization

Peter Strazdins*, Richard Alexander and David Barr,
The Jabberwocky Project,
Department of Computer Science at the
The Australian National University and
Alexander Technology

Workshop on XEN in High-Performance Cluster and Grid Computing
(The International Symposium on Parallel and Distributed Processing and Application 2006),
04 December 2006

(slides available from <http://cs.anu.edu.au/~Peter.Strazdins/seminars>)



THE AUSTRALIAN NATIONAL UNIVERSITY



1 Overview

- motivation
- background:
 - virtualization under Xen
 - TCP/IP stack processing under Linux SMP
- multiple GigE interface configurations using channel bonding and independent pathways
 - characteristics and implementation (including bypass of the Xen virtual machine monitor)
- experimental setup
- results
 - single-pair (single communication stream), 2-pair and 4-pair
- conclusions and future work

2 Motivation: SMP Clusters with Multiple GigE Interfaces

- low-end SMPs (esp. with multi-cores) are highly cost-effective for computational power
- communication performance of COTS clusters has not kept up
- many COTS motherboards support multiple I/O connections
 - e.g. IWILL DK8-HTX (Opteron) has three PCI-X / PCI buses and 14 device slots
 - adding network interface cards is at small relative cost
 - could these be used to balance the cluster's communication performance?
- (Gigabit) Ethernet (GigE) is the most widely used interface:
 - highly cost-effective; many application can run on it

3 Background: Virtualization under Xen

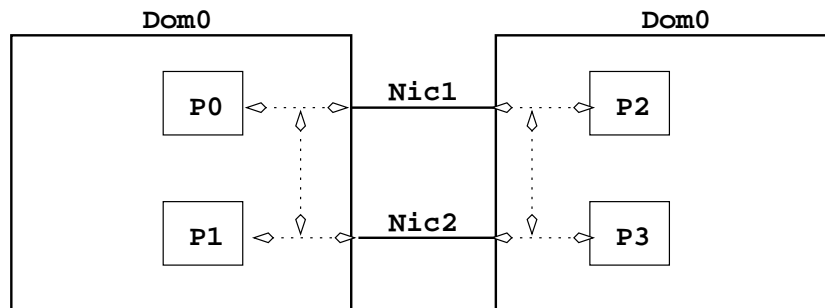
- virtualization offers many advantages (greater encapsulation), but typically comes at a significant performance penalty
- (OS) virtualization: a hypervisor (or virtual machine monitor, VMM) sits at a higher privilege level to the virtualized ('guest') OS
 - direct access to devices, page tables, privileged registers is by calls to the VMM
- Xen uses para-virtualization: the parts of the (Linux) kernel dealing with the above must be modified ('XenoLinux')
 - offers both potentially high performance and functionality
- requires one special guest OS (domain0, the 'driver domain'):
 - manages (configures, creates, destroys) a number of guest OSs (guest VMs)
 - external communication occurs through a virtual interface:
 - data is transferred by pseudo device drivers to domain0

4 Background: TCP/IP Stack Processing under Linux SMP

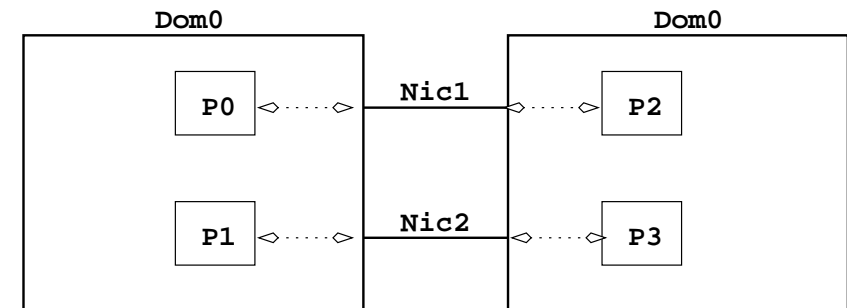
- Linux SMP 2.6 has sophisticated TCP/IP stack processing
- uses 2 kinds of locks: connection-related and socket-related (more frequently accessed)
- 2 broad strategies:
 - message-parallel: ||ize the processing (of different segments) of a single transmission
 - connection-parallel: ||ize processing of messages using different sockets (generally superior performance)
- recent studies have shown parallelization incurs some overhead (lock manipulation and cache pollution)
 - explains the generally poor performance of channel bonding

5 Communication Configurations with Multiple Interfaces

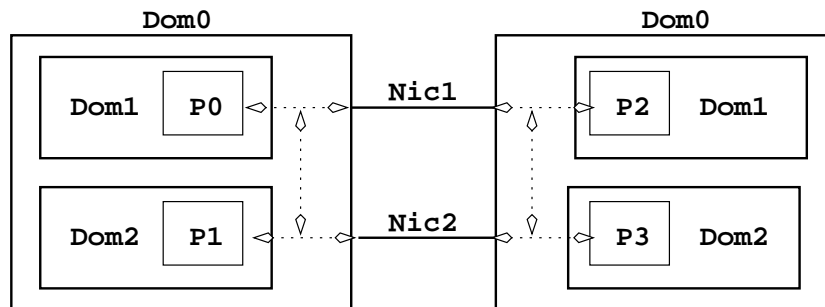
- consider aggregate bandwidth between MPI process pairs 0 & 2, 1 & 3:



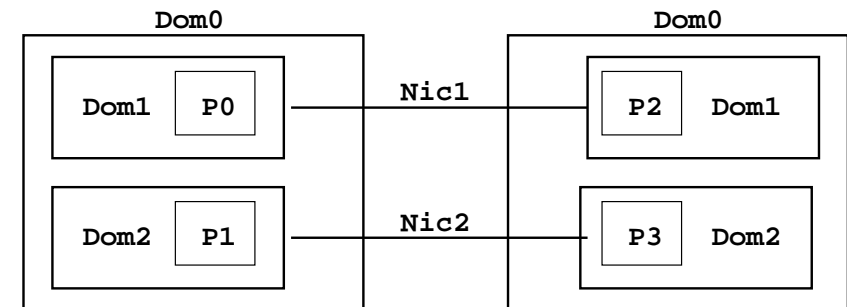
(a) driver.bond.2p



(b) driver.indep.2p



(c) guest.bond.2p



(d) guest-byp.indep.2p

- similarly have 1-pair (.1p - baseline) and 4-pair (.1p) configurations

6 Communication Configs: Characteristics & Implementation

- parallelization of TCP/IP stack: `*.bond: socket-level`, `driver.indep.*: connection-level`, `guest-byp.indep: full`
- communication bypass of the Xen VMM is achieved by modifying the guest VM's configuration file:
 - unset the `vif` variable (no virtual interfaces)
 - setting the `pci` variable to the desired bus/slot, e.g. `pci = ['03,04,0']`
 - the guest VM is bound to a particular CPU
- in `driver.indep`, the `route add` command is used to send messages to a particular process through the respective NIC
 - in the test programs, `sched_setaffinity()` used to bind processes to their respective CPUs
- in `indep.*`, interrupt requests for each NIC are bound to each CPU
 - with 4 NICs, found there were an insufficient number of IRQs; had to implement filesystems on a network bootable RAM disk!

7 Experimental Setup

- hardware:
 - 2 node cluster of dual SMP dual-core 2.2 GHz AMD Opterons
 - IWILL DK8-HTX_815 motherboard with 800 MB/s HyperTransport links
 - eth1: one of the in-built dual Intel GigE interfaces (PCI bus 3)
 - eth2, eth3: 82546GB dual Pro/1000 MT NIC (some TCP/IP offload, PCI bus 3)
 - eth4: 82541PI Pro/1000 MT NIC (PCI bus 1)
 - from `lspci`: all buses in 32-bit mode @ 66 MHz (264 MB/s)
- software:
 - use MPICH-2 MPI from OSUbench; MPI configured to use only eth1 to eth4
 - use multiple-pair version of OSUbench's latency (ping-pong), unidirectional and bi-directional bandwidth benchmarks

8 Results for Communication Configurations: Single Pair

- for `eth1`, uni-b/w 110 MB/s (c.f. GigE max. of 125 MB/s),
except `guest.bond`: only 50 MB/s !
- bi-b/w: 135 MB/s; `eth2/eth3`: 155 MB/s, `eth4`: 90 MB/s
- inter-node latency tests:

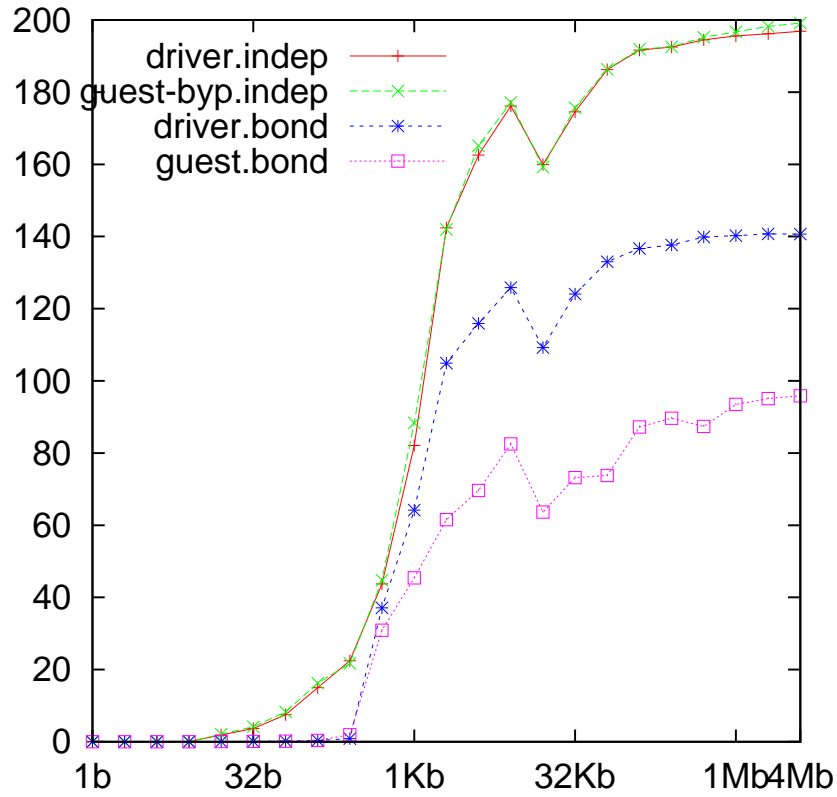
	*.indep	driver.bond	guest.bond
time at 1 B (μs)	87	91	106
B/W at 4 MB (MB/s)	107	105	51

- intra-node latency tests indicate `guest*.*` are using virtual interfaces ...

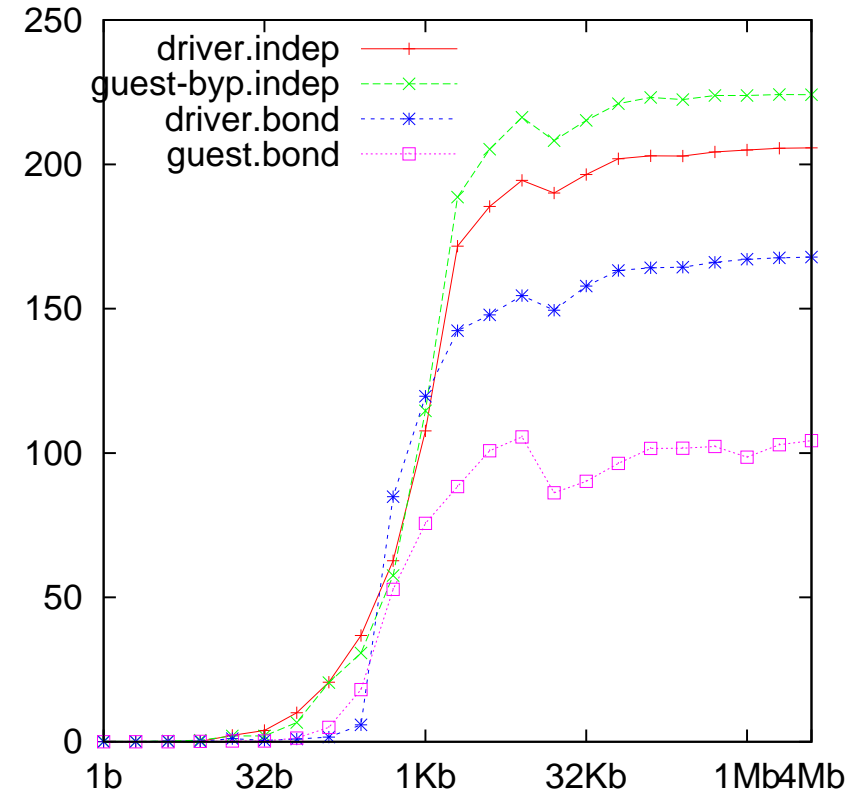
	driver.*	guest*.*
time at 1 B (μs)	18	30
B/W (MB/s) at 4 MB	530	50

- native Linux performance was essentially the same as `driver.indep`

9 Results: Bandwidth (MB/s) for 2-pair Configurations

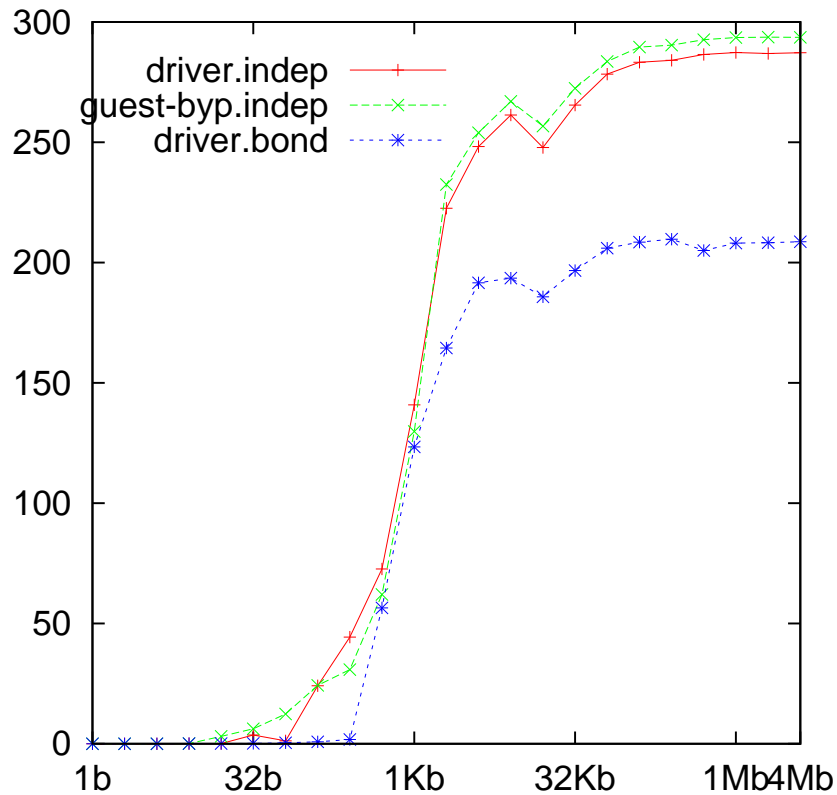


(a) uni-directional

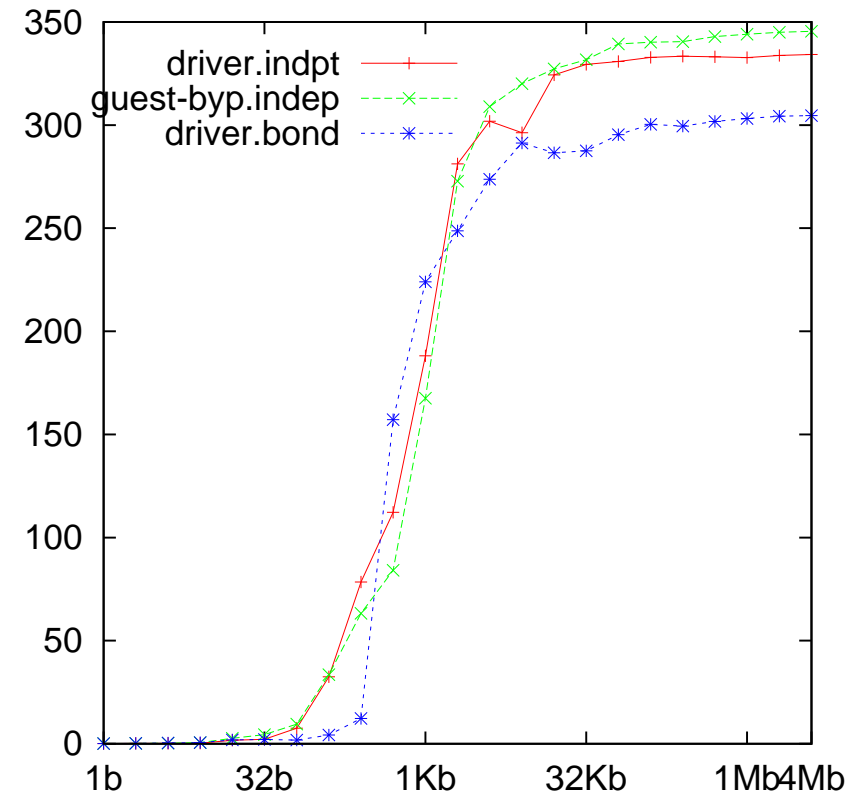


(b) bi-directional

10 Results: Bandwidth (MB/s) for 4-pair Configurations



(a) uni-directional



(b) bi-directional

11 Results for Multiple-Pair Configurations

- latency (at 4MB), in MB/s:

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

- native Linux (independent channels) behaved like driver.indep except for 4 pairs (latency slower, but uni-b/w up to 335 MB/s!)
- general observations:
 - large degree of variability in individual driver.indep.4p measurements; very small in guest-byp.indep
 - channel bonding performance increases with multiple communication streams
 - clear (although sub-linear) increase in bandwidth as network interfaces are added
 - likely diminishing return from adding yet more interfaces

12 Conclusions and Future Work

- worthwhile improvements in communication bandwidth can be achieved by using multiple network interfaces
- independent streams were generally faster significantly than channel bonding
 - best performance using Xen virtualized hosts with VMM-bypass due to full parallelization of TCP/IP stack processing
 - while not decisively faster, showed least variability in performance
- future work:
 - VMM-bypass needed for intra-node communication (non-trivial)
 - test application level performance
 - test with other motherboards and NICS; 8 CPU case?
 - develop virtual clusters: customization, migration
- virtualization offers many advantages for HPC; where VMM-bypass is acceptable, may even have benefits in performance !