The Wild West of UNIX I/O

Anil Madhavapeddy
University of Cambridge
twitter @avsm
http://anil.recoil.org

with Steven Hand, Richard Mortier, Malte Schwarzkopf,
Steven Smith, Chris Smowton, Robert Watson
IPC Matters!

Not just microkernels!
Data Processing
Cloud Computing
IPC Matters!

Not just microkernels!
Data Processing
Cloud Computing

Privilege Layers

JVM
CLR
KVM
VirtualBox
VMware
Xen

Multicore

Hyperthread
NUMA

CPUs
Impact on Throughput

Throughput (Gbs)

- AMD+Xen+4k+TCP
- AMD+Native+4K+Shmem

Throughput (Gbs)

- Intel I7+native+4k+Shmem
- Intel I7+native+4k+TCPnd

6x

10x
WHY   this matters
WHAT   one fix is
HOW   you can help
# Experiments

<table>
<thead>
<tr>
<th>48-core AMD</th>
<th>Linux 3.1 x86_64</th>
<th>Xen 4.1 x86_64</th>
<th>all tests in RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6</td>
<td>7 8 9 10 11 12</td>
<td>25 26 27 28 29 30</td>
<td>31 32 33 34 35 36</td>
</tr>
<tr>
<td>13 14 15 16 17 18</td>
<td>19 20 21 22 23 24</td>
<td>37 38 39 40 41 42</td>
<td>43 44 45 46 47 48</td>
</tr>
</tbody>
</table>
Experiments

<table>
<thead>
<tr>
<th>latency tests</th>
<th>between two CPUs</th>
<th>processes pinned</th>
<th>x/y grid of combinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>7 8 9</td>
<td>25 26 27</td>
<td>31 32 33</td>
</tr>
<tr>
<td>4 5 6</td>
<td>10 11 12</td>
<td>28 29 30</td>
<td>34 35 36</td>
</tr>
<tr>
<td>13 14 15</td>
<td>19 20 21</td>
<td>37 38 39</td>
<td>43 44 45</td>
</tr>
<tr>
<td>16 17 18</td>
<td>22 23 24</td>
<td>40 41 42</td>
<td>46 47 48</td>
</tr>
</tbody>
</table>

Experiments:
- latency tests between two CPUs:
- processes pinned:
- x/y grid of combinations:

Sunday, 5 February 12
## Experiments

<table>
<thead>
<tr>
<th>TCP</th>
<th>UDP</th>
<th>Shared memory</th>
<th>Pipes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>7 8 9</td>
<td>25 26 27</td>
<td>31 32 33</td>
</tr>
<tr>
<td>4 5 6</td>
<td>10 11 12</td>
<td>28 29 30</td>
<td>34 35 36</td>
</tr>
<tr>
<td>13 14 15</td>
<td>19 20 21</td>
<td>37 38 39</td>
<td>43 44 45</td>
</tr>
<tr>
<td>16 17 18</td>
<td>22 23 24</td>
<td>40 41 42</td>
<td>46 47 48</td>
</tr>
</tbody>
</table>
native, TCP

Latency in microseconds

Core ID

Latency in microseconds

Core ID

multiprocessor module

2 MPMs = 1 socket

inter-socket HyperTransport
Effect of virtualisation?

48-vCPU domain 0

Xen
Effect of virtualisation?

Native, TCP vs Xen-unpinned, TCP

Core ID vs Core ID

Latency in microseconds

Sunday, 5 February 12
Effect of virtualisation?

Xen dynamically remaps vCPU to pCPU mappings, hence no locality benefits.

Absolute latencies are also double.
Effect of virtualisation?
Effect of virtualisation?

![Graph showing latency comparison between native TCP and Xen-pinned TCP](image_url)
Effect of virtualisation?

Can pin vCPUs, but the enumeration is different from physical.

Also requires “root” access to the hypervisor, not a user-level operation.
Effect of NUMA?

Domain socket latencies are less “clear” than TCP but NUMA effects clear in native Linux.
Effect of NUMA+Xen?

Sunday, 5 February 12
Effect of NUMA+Xen?

All memory in Xen PV guest allocated from NUMA node 0.

Hence gets more expensive as we get more distant from socket 0.
Effect of NUMA+Xen?

WTF?

only seems to happen with UNIX sockets
Lower Bound: IPI

Inter-Process Interrupt (IPI) are used to wake up remote cores.

Modified Xen to self-test very early in boot, before any domains are started.

Play “spot the microarchitecture!”
Lower Bound: IPI
Hard to Predict!
WHY

WHAT

HOW

this matters

one fix is

you can help
Benchmark Suite

Hey, I’ll hack up a benchmark over the weekend to get the best method!

Easy, right?
Transport Mechanisms

Design a transport library that uses:

1) Pipes
2) Domain Sockets
3) TCP
4) Shared Memory
Process A | Process B
---|---

different address space

Kernel

Userspace

shared address space
Pipes

Process A  Process B

Kernel

Buffer  Buffer

 syscall  syscall

single copy  single copy

Userspace

App  App

zero copy

Buffer

syscall

single copy

App

Buffer

syscall

single copy

App

Buffer

syscall
Domain Sockets

vs pipes: bidirectional socket semantics (stream, datagram, credentials)

Kernel > syscall > single copy > App

Userspace > syscall > single copy > App

Sockets

vs pipes: bidirectional socket semantics (stream, datagram, credentials)
Shmem

Kernel

Userspace

Process A

App

Shared Buffer

Process B

App

<table>
<thead>
<tr>
<th>next cons</th>
<th>Shared Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>next prod</td>
</tr>
</tbody>
</table>

- **header**
- **size**
- **flags** (ready/stop/waiting)
- **payload**

**how to sync with no kernel syscall?**
Shmem
spin
unsafe

Kernel

Userspace

Shared Buffer

App

Spin

App

Spin
The diagram illustrates the use of shared memory (Shmem) in a process and the impact of spinning in userspace and the kernel. The key points are:

- **Process A** and **Process B** are connected through a **Shared Buffer**.
- Spinning in **Process A** and **Process B** is shown to be **unsafe**.
- **100% CPU** is always present in the system, indicating constant CPU usage.
- The diagram highlights the flow of applications (App) and the use of shared memory (Shmem) in both processes.
Shmem
spin
unsafe

Kernel

Userspace

100% CPU always

App

Spin

Shared Buffer

App

Spin

very low latency
Shmem spin unsafe

100% CPU always

very low latency

horrible if virtualised

Shared Buffer

Process A

Process B

Kernel

Userspace

App

Spin

App

Spin
Shmem
spin
safe

Kernel

Userspace

Process A

Process B

Shared Buffer

Private Buffer

App

Spin

App

Spin

private copy to not “trust” other side
Transport API Design

• POSIX API bakes in trust assumptions, and decouples names versus addresses.

• At least one data copy is required to go from kernel to userspace: not zero-copy.

• Establishing shared memory segments defined by POSIX, but no standard equivalent of a shared-memory “sockets”.
Throughput

http://www.cl.cam.ac.uk/netos/ipc-bench

Throughput [Mbps]

(a)

(b)

(c)

(d)

same core

off core

same socket

near socket

far socket
WHY this matters
WHAT one fix is
HOW you can help
ISA (AMD/Intel) → Layout (ACPI SRAT/SLIT) → NUMA (mem controller)

Native/Virtualised → Hypervisor (EPT/NPT, HVM/PV)

OS/Kernel → Synchronisation (futex, mutex, syscall ...)

Flow Control → Congestion Control

Trust → Transport Security (SSL, TCPrypt, SSH)

Buffer Sizes → Request Sizes

Data Transformation → Language Runtime

Hardware

Virtualisation

OS

Channel

Trust

Data Pattern

Processing
Benchmark Suite

- IPCbench is open-source
  - [http://github.com/avsm/ipc-bench](http://github.com/avsm/ipc-bench)
- BUT a work in progress:
  - Linux only, not packaged up
  - Documentation
  - We don’t have much hardware.
Moar data needed!

- Significant effects of Xen and NUMA, odd cache effects between safe/unsafe transports: sometimes zero-copy wins, other times not.

- **Microarchitectural variance** is large (Intel QPI vs AMD Hypertransport). 100Gbs+ in best QPI, AMDs max out at 30Gbs often. AMD Bulldozer?

- Hyperthreads linger like a bad smell: worse than talking to a different core, or to the same core without hyperthreads, and introduce variance.

- Some techniques are awful in particular conditions: e.g. vmsplice on VirtualBox is dramatically bad.

- [http://www.cl.cam.ac.uk/netos/ipc-bench](http://www.cl.cam.ac.uk/netos/ipc-bench)
Long Term Data

• Collect long-term performance data on this as architectures and OS’s evolve.

• Use Git for DVCS, Github for hosting

• Challenges: versioning, compression, provenance, multi-variate analysis.

• Pros: a community performance database would be invaluable.

• http://www.cl.cam.ac.uk/netos/ipc-bench
• Can we all agree that this is too complex for the average (or even awesome) programmer?

• Time for **first-class I/O channels**!

• **FABLE** is our ongoing attempt at a “new” sockets API for high-performance data.

**Persistent “flows”**

**Dynamic reconfiguration**

**Automatic transport choice**

**Xen/Kernel/Userspace**

---

Reconfigurable I/O

In the cloud, no-one can hear you stream (or live migrate).

- hence, we always use TCP between VMs.

TCP (sloooow)

Web <-> DB

Xen 1 <-> Xen 2
Reconfigurable I/O

In the cloud, no-one can hear you stream (or live migrate).

- hence, we always use TCP between VMs.
Reconfigurable I/O

In the cloud, no-one can hear you stream (or live migrate).

- hence, we always use TCP between VMs.

libvchan
(2-4x faster)

Web ↔ DB

Xen 1

Xen 2
Reconfigurable I/O

In the cloud, no-one can hear you stream (or live migrate).

- hence, we always use TCP between VMs.
Reconfigurable I/O

In the cloud, no-one can hear you stream (or live migrate).

• hence, we always use TCP between VMs.

libvchan (shmем)
reconfigure channel!
TCP (+SSL?)

Web
Xen 1

TCP
Xen 2

DB
Reconfigurable I/O

name daemon

flow manager

proc

proc

proc

proc

kernel
Reconfigurable I/O
Reconfigurable I/O

name
daemon
flow
manager
# Fable API

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xio_register_name</code></td>
<td>Register process or VM</td>
</tr>
<tr>
<td><code>xio_connect</code></td>
<td>Connect to remote URL</td>
</tr>
<tr>
<td><code>xio_getreadbuf</code></td>
<td>Obtain a read buffer</td>
</tr>
<tr>
<td><code>xio_getwritebuf</code></td>
<td>Obtain a write buffer</td>
</tr>
<tr>
<td><code>xio_release</code></td>
<td>Free a read buffer</td>
</tr>
<tr>
<td><code>xio_commit</code></td>
<td>Commit a write buffer</td>
</tr>
<tr>
<td><code>xio_poll</code></td>
<td>Poll for XIO events</td>
</tr>
</tbody>
</table>
FABLE connection

handle “foo” “bar”  process pid  VM uid
connect

dynamically determine best transport mechanism between “foo” and “bar”.

shared memory+futex
...or TCP/IP
...or multipath TCP!
FABLE connection

- handle
- “foo” “bar”
- process pid
- VM uid
- connect
- shmem context
- TCP/IP DMA
- getbuf
- shmem buffer
- TCP/IP buffer
- commit
- futex
- free
- csum offload
- page-aligned, with reserved header space

in-place shmem buffers
FABLE uses

- **Improve cloud I/O**
  - Upgrading the Xen ring API
  - CIEL/Hadoop for Big Data ([http://ciel.io](http://ciel.io))

- **Microkernel support**
  - MirageOS ([http://openmirage.org/](http://openmirage.org/))
  - “Pick your favourite language”

- **Reconfigurable hardware!**
Conclusion

• IO is increasingly slow due to NUMA + virtualisation layers.

• IPCbench can help gather long-term performance data for the community.

• FABLE is an ongoing project to add first-class flows to UNIX-like operating systems.

http://www.cl.cam.ac.uk/netos/ipc-bench