Performance Optimisations for HPC workloads

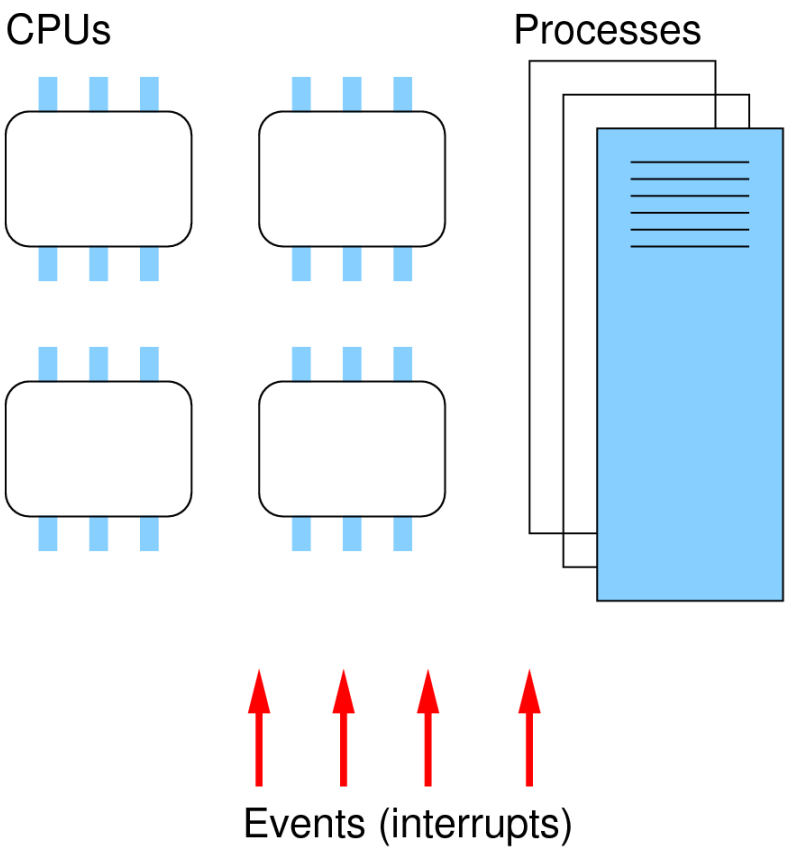
August 2008

Imed Chihi

Agenda

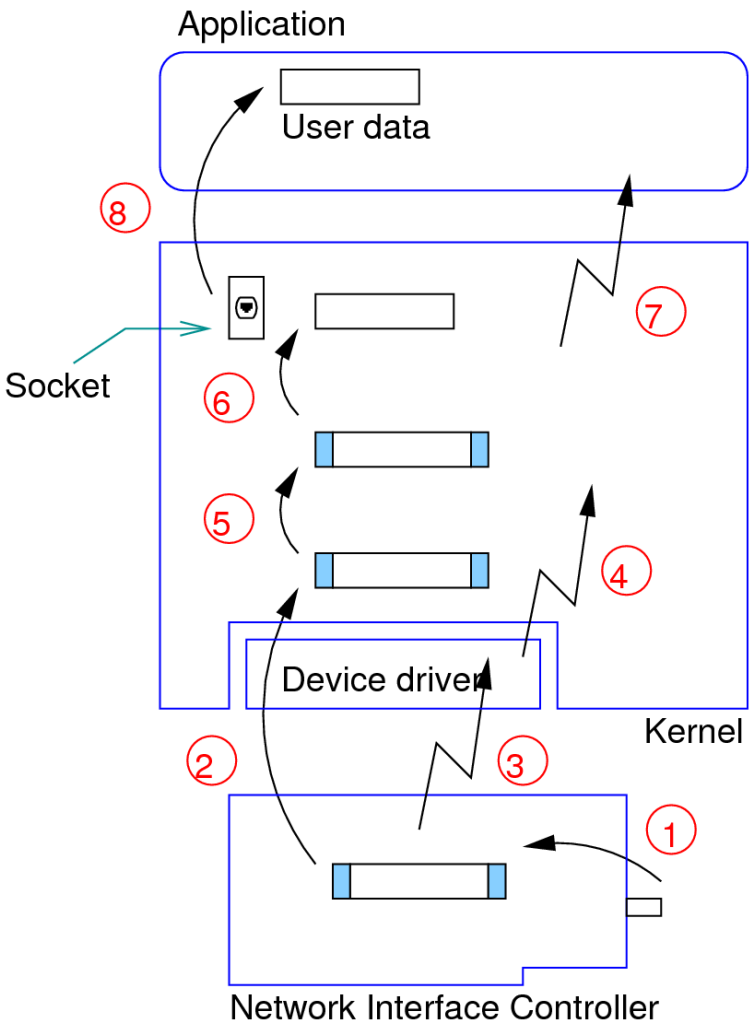
- The computing model
- The assignment problem
- CPU sets
- Priorities
- Disk IO optimisations
- gettimeofday()
- Disabling services
- Memory management and swapping
- Network tuning

The computing model



- Processes are contexts to get work done
- Processes execute on CPUs
- Events may change the course of executions
- Events are handled by CPUs

Interrupt handling – packet receive



(1) NIC receive PDU from the network

(2) NIC copies PDU into kernel buffers using DMA

(3) NIC notifies kernel about the received PDU by raising an interrupt

(4) Device driver acknowledges the interrupt and schedules a *softirq*

(5) Kernel hands the PDU up to the IP stack for routing decisions

(6) Kernel extracts the data and copies it to the corresponding socket buffer

(7) Kernel notifies all processes waiting on the socket

(8) The `read()` call issued by the application proceeds and data is copied into the application buffers

Interrupt handling

- Interrupts are handled by an interrupt controller: Advanced Programmable Interrupt Controller (APIC)
- NIC raises a hard interrupt at each event: packet reception or transmission
- Hard interrupt handling cannot be interrupted, may lockup the system if it sleeps or takes too long to complete
- Hard interrupt handlers perform minimal work and schedule the remainder of the job to be handled asynchronously by a softirq
- Softirqs are processed as regular kernel code by special kernel threads: `ksoftirqd/`X

Interrupt handling

- Kernel will drop packets if it cannot pick them from the NIC quickly enough
- Hard interrupts are reports in: `/proc/interrupts`
- `irqbalance` balances hard interrupts across CPUs
- Hard interrupt cost can be mitigated by interrupt coalescing

The assignment problem

- Optimise the assignment of processes and Interrupts handling to CPUs
- Process CPU affinity
  - `taskset`
  - `sched_setaffinity()`
  - `isolcpus` boot parameter
- IRQ CPU affinity
  - `irqbalance`
  - `/proc/irq/<irq#>/smp_affinity`
- Process memory affinity

cpusets

- Most useful on large systems
- Partitions the system memory and CPUs
- Optimises application performance by pinning them to CPUs
- Useful for web server and database workload for instance
- Extended framework to implement process affinity to CPUs and memory
- Can be tested with fake NUMA on non-NUMA systems
  - Boot with: `numa=fake=4*256`

cpusets – example

```
# mount -t cgroup -ocpuset cpuset /dev/cpuset
# cd /dev/cpuset
# mkdir hpc
# cd hpc
# /bin/echo 2-3 > cpus
# /bin/echo 1 > mems
# /bin/echo <some_pid> > tasks
# cat /proc/<some_pid>/cpuset
```

Process priorities

- Normal priorities
  - Can be set with nice and renice
  - Range from -19 to +20
  - Scheduling class: `SCHED_NORMAL`
- Realtime priorities
  - Can be set with `chrt` and `sched_setscheduler()`
  - Range from 1 to 99
  - Scheduling class: `SCHED_FF` and `SCHED_RR`

Disk IO tuning

- Choosing an IO scheduler
  - `echo cfq > /sys/block/<device>/queue/scheduler`
- Optimise reads, not writes
- Read/write in bulk when possible
- Adjust request queue length
  - `/sys/block/<device>/queue/nr_requests`
- Maximum read-ahead
  - `/sys/block/<device>/queue/read_ahead_kb`
- Mount with `-o noatime`

Time keeping

- Hardware timers
  - RTC
  - PIC, APIC
  - HPET
  - TSC
  - PMTIMER
- Accelerated gettimeofday()
- Adjust the duration of a tick
  - Default is 1 ms (1 KHz clock)
  - `tick_divider=2..10`

Disabling services

- All desktop workloads
- cron
- cpufreq
- irqbalance
- gpm
-

Memory management

- Memory allocation remains very undeterministic
- Memory allocator “prefers” real time tasks
- Avoid repeated allocations
- Moderate over commit
- On NUMA systems
  - Use numactl for process memory affinity
  - cpusets still override numactl

Lab

Memory usage in matrices multiply

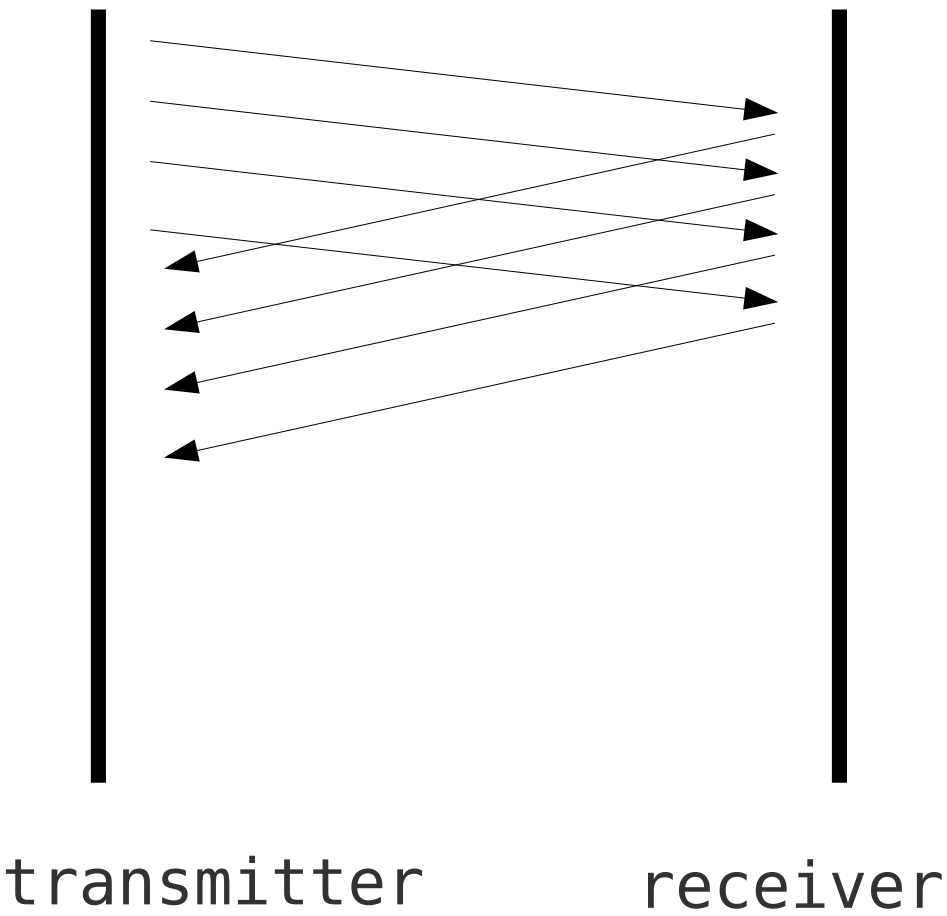
Optimising a large build job

Optimising kernel buffers

- Used for DMA transfers from NIC, socket buffers, fragmentation and re-assembly
- Buffers consume the low memory zone which might be problematic on IA32 architectures
- TCP reports its buffer size to the sender in order to enforce a flow control
- Can be adjusted by:

```
net.core.rmem_default
net.core.rmem_max
net.core.wmem_default
net.core.wmem_max
```

The bandwidth-delay product



- If the sender waits until an ACK is received for each packet sent, then the transmission will be inefficient
- Transmitter anticipates the reception of ACKs and sends multiple packets
- The transmitter needs to keep a copy of the packet until it receives an ACK for it. It might be lost
- The receiver reports the size of its buffer (window) with each ACK
- Before receiving the first ACK, the transmitter can send bandwidth*RTT

Slow start implications

- In early days of the Internet packet loss usually meant a packet corruption due to bad links quality, the obvious remedy was to retransmit
- Modern networks are more reliable and packet loss usually means a congestion at the routers. The old “remedy” of retransmission actually worsens the problem
- Modern TCP implementations would severely drop their transmission rates when they detect a packet loss assuming a congestion occurred
- This makes correct tuning of buffers even more important to networking performance
- tcp_slow_start_after_idle

TCP-specific buffers (window)

- TCP automatically adjusts the size of its *sliding window* with `net.ipv4.tcp_window_scaling`
- `TCP buffers controlled by:`
  - Overall TCP memory in pages: `net.ipv4.tcp_mem`
  - Input/reader in Bytes: `net.ipv4.tcp_rmem`
  - Output/writer in Bytes: `net.ipv4.tcp_wmem`

Fragmentation and re-assembly

- Most payloads of PDUs exceed the MTU of the underlying physical network
- Path MTU (lowest MTU across the path links) can be “discovered”
- NFS for instance transmits at least 8 KB packets which are larger than the 1500-bytes Ethernet MTU
- Fragmentation buffers adjusted using
  - Minimum size: `net.ipv4.ipfrag_low_thresh`
  - Maximum size: `net.ipv4.ipfrag_high_thresh`
  - Expiration time for fragments: `net.ipv4.ipfrag_time`

Fragmentation/re-assembly statistics

- Summary statistics

```
# netstat -s
```

- Reassembly failures

```
# cat /proc/net/snmp | grep '^Ip:' | cut -f17
-d' '
```

- Reassembly failures indicate a need to tune buffers
- NFS and Denial of Service attacks are common causes of high reassembly counts
- Enable Jumbo frames on Gigabit Ethernet when applicable

TCP segmentation offloading

- TCP segmentation and reassembly operations are quite expensive
- Some NICs implement the TCP reassembly logic in hardware
- Check status with:

```
# ethtool -k eth0
```

- Enable with:

```
# ethtool -K eth0 tso on
```

- The same applies to receive and transmit checksumming

TCP overhead