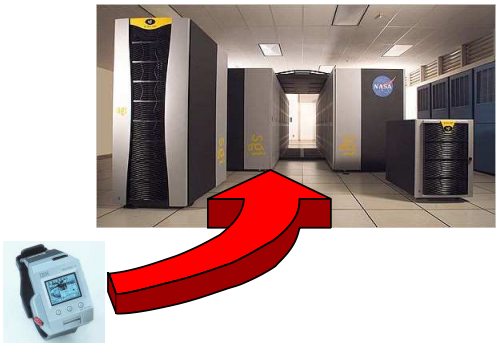
SCALABILITY, I/O AND ALL THAT
AOS 2009.10.16

Peter Chubb
peter.chubb@nicta.com.au



Scalability

- Essentially the same kernel across wide range of hardware

  or not?
- Performance scales with hardware

  or does it?

Same kernel source, same core algorithms (in general); conditional compilation does change some features.

Phase Changes

- Processor change
- Uni → SMP
- UP → SMT

- NUMA

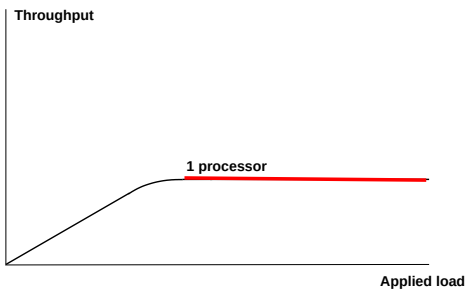
- Cluster

Scalability Issues

- Number of Processors
- Number of Spindles
- Amount of Memory
- Power Consumption...
- Scalability depends on Workload

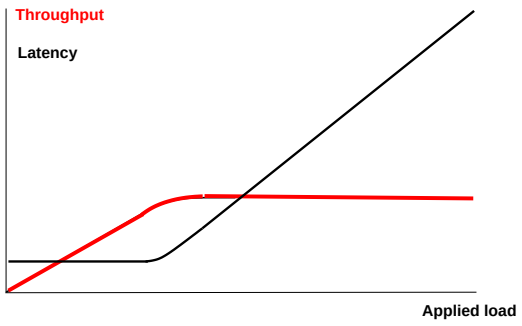
Ideal Scalability

System Performance



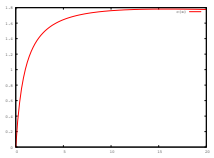
The ideal system performance curve has three parts. At the left, throughput is detemined by the rate at which work requests arrive; there is little or no queueing, and latency is governed by how long a job takes to do. In the middle, where the curve is, jobs sometimes arrive while a previous job is still running. Latency is governed by both the time a job takes, and the time it spends queued – which depends on how long previously queued jobs take. At the right of the curve, latency is determined almost entirely by queuing time; and throughput is determined solely by the time the server takes to do a job. In this state, one or more resources on the server are bottlenecked; removing the bottleneck will move the curve up and the bent part of the curve to the right.

System Performance



System Performance

Gunther's law:



$$C(d) = \frac{d}{1 + \alpha(d-1) + \beta d(d-1)}$$

where:

d is demand

α is the amount of serialisation: represents Amdahl's law

β is the contention in the system.

Scalability Considerations

- On-chip cache (ns)
- local L2 cache (10s of ns)
- Local RAM (100s of ns)
- Other processor's cache (μs)
- Disc backing store (10s of ms)

Scalability Considerations

Other factors

- Locking 'grain'
- Interrupts — to one processor or to any?
- Disc and other I/O parallelism
- Cache fights
- TLB contention

NUMA

- Non-Uniform Memory Architecture
- Processors joined by a high speed network.
- Typically use hypercube or bristled hypercube topologies
- Nodes may be SMP (Compaq) or use crossbar (SGI)
- ccNUMA (cache-coherent NUMA) puts DSM in hardware.

Goals of NUMA

1. Better performance at lower cost (two processors are cheaper than one twice-performance processor — and when the twice-performance processor is developed it can be plugged into the earlier architecture)
2. Hide architecture from user-mode programs
3. Scalable performance enhancement

NUMA concepts

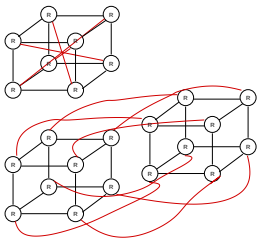
RAD the Resource-affinity-domain, (aka 'Cell')

SMP Symmetric Multi-Processing — All processors have equal status

Master-Slave Some system calls, interrupts must run on Master processor.

NUMA concepts

Hypercube etc

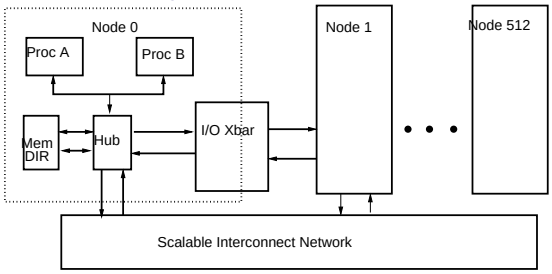


NUMA CONCEPTS

- 'Bristled' hypercube (e.g., SGI Altix).
- Plain hypercube (e.g., Compaq).
- $O(\sqrt[3]{n})$ hops for n nodes.
- Each node has local memory.
- Some nodes have I/O (e.g., Disc, framebuffer, serial, parallel, etc)
- Clocks may be synchronised with NTP, or be delivered from a central master.
- Nodes may not be exactly homogeneous.

NUMA CONCEPTS

Origin 2000 Node architecture



MEMORY HIERARCHY RE-VISITED

- On-chip cache (ns)
- local L2 cache (10s of ns)
- Local RAM (100s of ns)
- Other local processor's cache (μs)
- **Remote processor's RAM or cache (ms)**
- Disc backing store (10s of ms)

PROCESS STATE

Processes have state

- memory footprint
- parent
- PID
- queued signals
- open files
- network endpoints

PROCESS STATE

PIDs

- Want a single global PID pool
- Can manage in two ways
  - Single pool-manager, used by all, or
  - Use RAD ID as part of PID.
- Either way need a globally-consistent mapping pid→proc struct

SCHEDULING AND DISPATCHING

- Memory switching is expensive
- Therefore try to schedule memory-sharing threads at same time.
- Two approaches
  1. Gang Scheduling
  2. Job models

Scheduling and Dispatching

Dispatching

- Per-processor run queue
- Thread queued to same processor as last time
- Thread runs in same cell as last time, if possible.
- Otherwise either goes onto queue for idle processor, or onto global queue.
- At dispatch time, threads taken from local pushback queue, or per-processor queue or global queue.

Scheduling and Dispatching

Gang Scheduling

- Threads run in a memory space
- Schedule memory spaces, let threads fend for themselves (RR or FIFO depending on user selection)
- Try to spread threads over 'nearby' cells.

Scheduling and Dispatching

Job Currency scheduling

- SGI/Cray Irix concept
- 'Job' is set of threads+processes that are scheduled together.
- Each job given a 'weight' in proportion to priority
- At regular intervals, time is handed out to jobs in proportion to weights
- Jobs divided into 'over entitlement' and 'under entitlement'
- Threads belonging to under-entitlement jobs have

Scheduling and Dispatching

higher pri than the rest: queue resorted on every scheduling interval

- Threads round robin within priority band

Scheduling and Dispatching

Job Currency Scheduling (cont)

- Scales to a few hundred processors only!
- Not as good as gang for many purposes
- Needs combination with topology scheduler.

General Principles

- Locality of reference (use per-processor data area where appropriate)
- Replicate read-only data, including user Text segment for MT progs;
- use 'const' in kernel code; linker support for replicated RO-sections.
- Try to move shared elements 'close' to users; stripe shared memory.
- Use fine-grain MR-locks where appropriate
- Use fetch-and-op operations rather than locked sections

General Principles

where possible.

Cache-Friendly Data Layout

- Group data in cache blocks by usage
- keep data consumers to the minimum
  - per thread
  - per processor
  - per proc
  - per RAD
  - Replicate read-mostly data

More general guidelines

- Keep differently locked-data in different cache blocks
- Group read-mostly data into a single cache-block
- Put read-write data in the same cache block as its lock
- Group data according to how frequently it is accessed
- Minimise number of cache blocks for most frequent code paths.

Topology Scheduling

(e.g., SGI's 'Miser' program, Compaq's Processor Sets)

- Reserve RADs for particular job(s)
- Can reserve nearby memory in other RADs as well if nec.
- (pinning, processor sets)

Some Conclusions

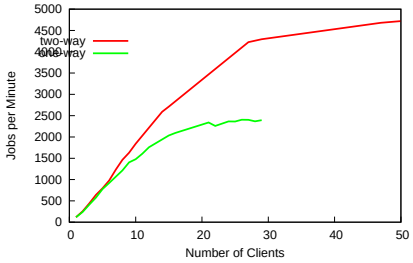
- Can get scalable performance out of ccNUMA
- Needs careful attention to detail in OS
- Tuning for one architecture can break others.

References

- 'The SGI Origin: A ccNUMA Highly Scalable Server', http://www.sgi.com/origin/images/isca.pdf
- Schimmel, 'Unix Systems for Modern Architectures'

Scalability in Practice

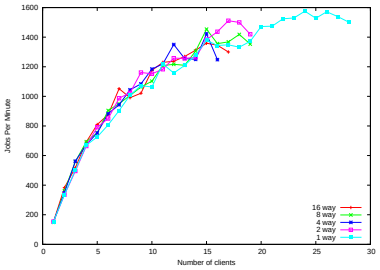
RX2600, OSDL's reimplementation of AIM7



On an HP RX2600 with two processors, the OSDL Aim7 benchmark is perfectly scalable.

Scalability in Practice

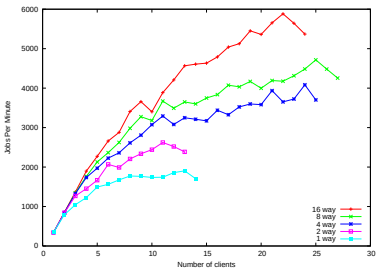
Altix, single spindle



But on the Altix, the scalability is awful. What's going wrong? We don't have access as root (borrowed machine from UWaterloo), and may not install our own instrumented kernels to see what's going on, so let's speculate that there's an I/O problem (as we know the benchmark scales with processors on an SMP machine), and try with a faster disc:

Scalability in Practice

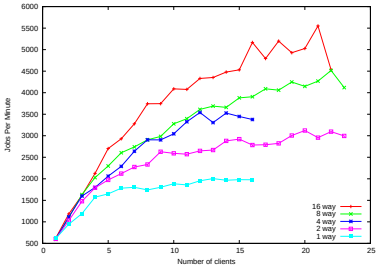
Altix, RAID



Better, but we still don't see the near-perfect scalability we expect. So try with a ram-disk:

Scalability in Practice

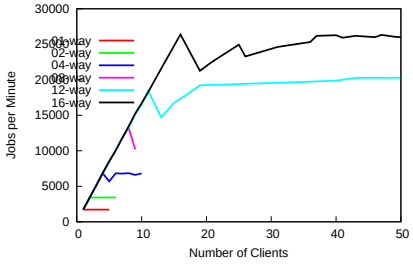
Altix, RAM disk, ext3



Even here there is a problem. It looks like there's a major problem in the I/O subsystem on a NUMA machine in Linux 2.4.21-sgi.

Scalability in Practice

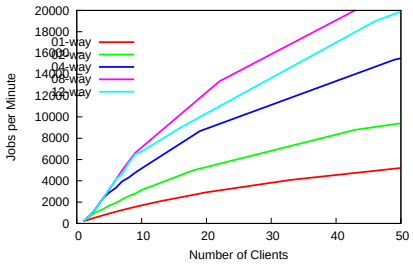
2.6, HP Olympia, CPU-only load



So let's try 2.6.11, on an HP Olympia with 12 processors spread across 4 nodes. Again, a CPU-intensive, no Disk I/O workload shows reasonable scalability with number of processors.

Scalability in Practice

Disk Intensive workload, tmpfs



This shows very poor scalability. The 12-way actually performs worse than the 8-way.

Scalability in Practice

Analysis

- Microstate Accounting
- `q-syscollect`
- Lock Metering

Microstate accounting showed much time on the CPU; `q-syscollect` showed much time in `ia64_spinlock_contention`, so we ran with lockmetering:

Scalability in Practice

```
SPINLOCKS          HOLD            WAIT

UTIL   CON    MEAN( MAX )   MEAN( MAX )(% CPU)     TOTAL NOWAIT SPIN RJECT  NAME

72.3% 13.1% 0.5us(9.5us)   29us( 20ms)(42.5%)  50542055 86.9% 13.1%  0%
find_lock_page+0x30

0.01% 85.3% 1.7us(6.2us)   46us(4016us)(0.01%)      1113 14.7% 85.3%  0%
find_lock_page+0x130
```

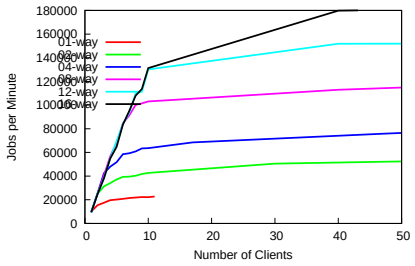

Scalability in Practice

```
struct page *find_lock_page(struct address_space *mapping,
                                    unsigned long offset)
{
        struct page *page;
        spin_lock_irq(&mapping->tree_lock);
repeat:
        page = radix_tree_lookup(&mapping>page_tree, offset);
        if (page) {
                page_cache_get(page);
                if (TestSetPageLocked(page)) {
                        spin_unlock_irq(&mapping>tree_lock);
                        lock_page(page);
                        spin_lock_irq(&mapping>tree_lock);
                                        ...
```


So let's replace the spinlock with a rwlock...

Scalability in Practice

Disk-intensive workload, tmpfs, rwlock



And bingo, it scales much better.

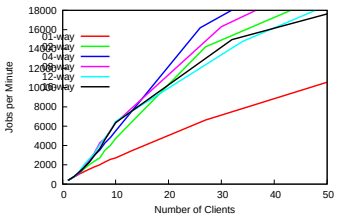
Scalability in Practice

- Slope still positive
  - Spare capacity available
  - **Latency** a problem

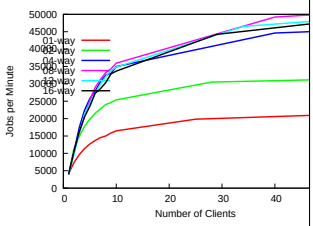
Scalability in Practice

Other File Systems On Ram disk

ext3

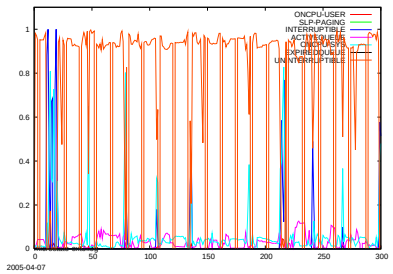


xfs



Scalability in Practice

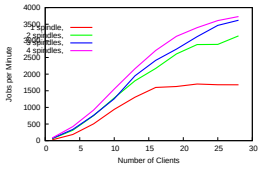
Microstate accounting output, ext3



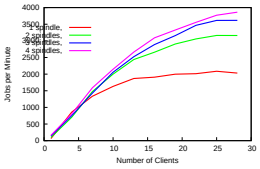
Scalability in Practice

Scalability by spindles

ext3



xfs



Tackling scalability problems

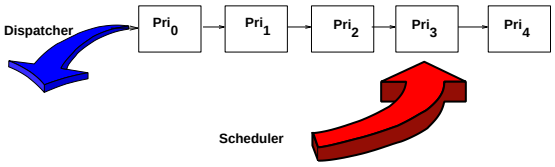
- Find the bottleneck
- fix or work around it
- check performance doesn't suffer too much on the low end.
- Experiment with different algorithms

Interrupts

- ➜ Steal time from current process
- ➜ Therefore can cause negative scalability.

Interrupts

Schedulers and Dispatchers



Scheduler Classes

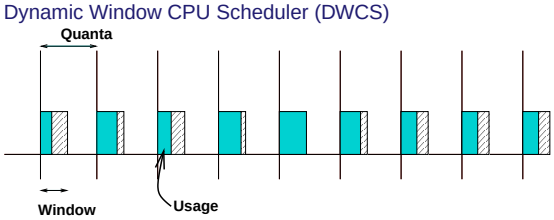
- Real-Time
- Time share schedulers
- Entitlement Schedulers
- Batch Schedulers

Isochronous Schedulers

- For Multi-Media apps
- Guarantee a proportion of the processor.

Isochronous Schedulers

Dynamic Window CPU Scheduler (DWCS)



Entitlement Schedulers

- Aim to share resources in proportion to *entitlement*
- Shares
- Or proportions (must be $< 100\%$)

Entitlement Schedulers

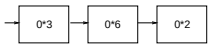
- Feed-forward (e.g. Lottery scheduler)
- Feed-Back (e.g., Kay & Lauder 1988)

Shares

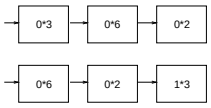
- Allow entitlement holders to come and go.
- Each *user* is given shares
- Entitlement = $S_u/\Sigma_i S_i$
- where i runs over all active *users*.

Simple Entitlement Scheduler

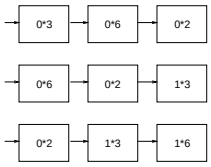
Simple Entitlement Scheduler



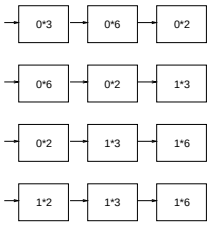
Simple Entitlement Scheduler



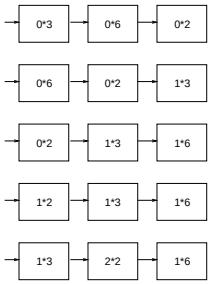
Simple Entitlement Scheduler



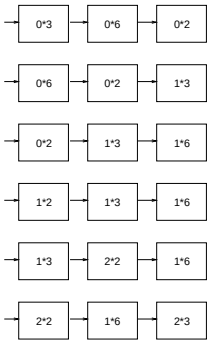
Simple Entitlement Scheduler



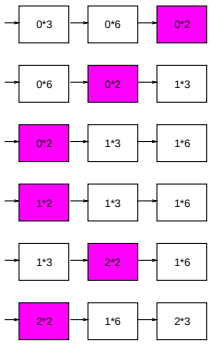
Simple Entitlement Scheduler



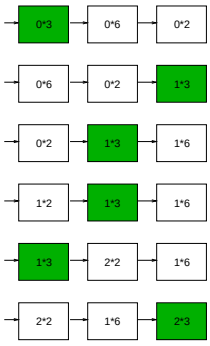
Simple Entitlement Scheduler



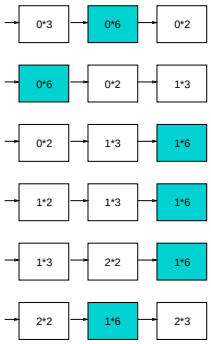
Simple Entitlement Scheduler



Simple Entitlement Scheduler



Simple Entitlement Scheduler



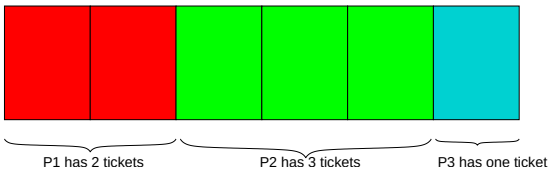
Problems

- Infinities.
- Processes that sleep.
- Processes too low level.

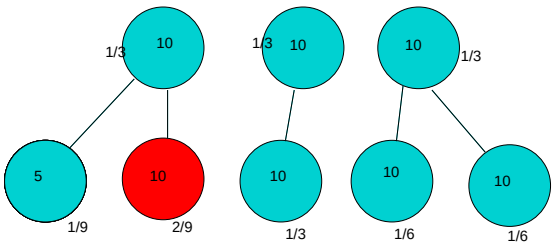
Lottery Schedulers

- Each entity has some tickets
- Randomly select a ticket at each decision point
- Entities get time in proportion to how many tickets they hold.

Lottery Schedulers



Inactive Entity Problem



Inactive Entity Problem

- Need feedback loop
- Boost under-utilising processes
- Slow down over-utilising processes
- Can lead to starvation

WILLIAMS 2004

- Use Kalman estimator as usage
- $u_i = \alpha u_{i-1} + (1-\alpha)\rho$
  where the half life of the system is $\frac{\ln 0.5}{\ln \alpha}$
- Can update once after multiple time steps:

$$u_m = \alpha^{m-i} u_i + (1-\alpha^n) + (1-\alpha)\rho$$

WILLIAMS 2004

- Priority is then $\frac{u}{\epsilon}$ where ϵ is entitlement.
- Hierarchical possible: $\epsilon = \epsilon_u \epsilon_p \epsilon_t$ providing
  $\forall x \in u, p, t : \sum_x \epsilon_x = 1$

WILLIAMS 2004

results:

- Patent granted
- Forms basis of ARMtech (used in Citrix).
- Earlier version in Solaris Resource Manager
- Free version was posted to LKML.
- Works very well, with low overhead.s

DISKS AND FILE SYSTEMS

- heads, tracks, sectors
- seek is **very expensive**