Isolating Cluster Jobs for Performance and Predictability

Brooks Davis <brooks@aero.org>
Enterprise Information Systems
The Aerospace Corporation

BSDCan 2009
Ottawa, Canada
May 8-9, 2009

The Aerospace Corporation

Who We Are

- Since 1960 The Aerospace Corporation (Aerospace) has operated a federally funded research and development center (FFRDC) in support of national-security, civil and commercial space programs.
  - *The Aerospace FFRDC provides scientific and engineering support for launch, space, and related ground systems*
  - *It also provides the specialized facilities and continuity of effort required for programs that often take decades to complete.*
- The FFRDC's core competencies
  - *launch certification*
  - *system-of-systems engineering*
  - *systems development and acquisition*
  - *process implementation*
  - *technology application*

Breadth and depth of technical and programmatic expertise

•The Aerospace Corporation (cont.)

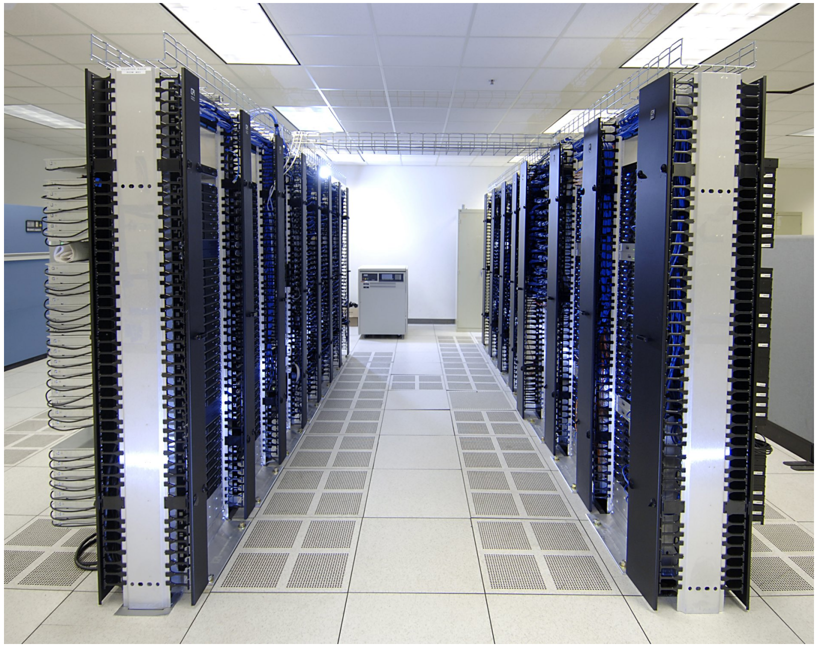
- Over 2400 engineers
  - *Virtually every discipline represented and applied*
- Vast problem space
  - *Everything related to space*
- Engineering support applications of all sizes
  - *From small spreadsheets*
  - *...to large traditional applications*
  - *...and large parallel applications*

A large and complex user and application base

•The Fellowship Cluster

HPC at The Aerospace Corporation

- 352 dual-processor nodes
  - *1392 cores*
- Gigabit Ethernet network
  - *10Gbps for switch-switch and storage links*
- FreeBSD 6.x i386
  - *Planning a move to 7.1 amd64*
- Sun Grid Engine scheduler
- ~40TB of NFS storage
  - *Sun x4500*
- Other resources
  - *Two smaller clusters coming soon*
  - *Some large SMP systems*



The Aerospace Corporation's primary HPC resource since 2002

Outline

The Rest of the Talk

- The Case for Resource Sharing
- The Trouble With Sharing
- Interesting Sharing Issues
- Some Possible Solutions
  - *Whole Node (or Larger) Allocations*
  - *Gang Scheduling*
  - *Single Application (Sub-)Clusters*
  - *Virtualization*
  - *Virtual Private Servers*
  - *Resource Limits and Partitions*
- Our Experiments
  - *SGE Shepherd Wrapper*
  - *Memory Backed Temporary Directories*
  - *Variant Symbolic Links*
  - *CPU Set Allocator*
- Conclusions and Future Work

The Case for Resource Sharing

Efficient Use of Scarce Resources

- Unique resources like The Fellowship Cluster need to be shared
  - *Users need things at different times*
  - *We can not afford to buy cluster for each user*
    - Even if they could use it all the time, we could not afford to administer all of them
- Users demand quick access to partial results
  - *If we block one user completely while another uses resources inefficiently we increase the time to a partial solution*

Sharing is required for efficiency

The Trouble With Sharing

Contention Leads to Increased Overhead

- Resource contention happens
  - *Users sometimes need the same thing at the same time*
  - *Some jobs use more resources than they request*
- Contention causes problems with performance
  - *Job completion time is difficult to predict in the face of contention*
  - *Sufficient contention raises OS overhead*
    - Mostly due to context switching and swapping
    - Some due to queue overruns

Sharing is required for efficiency, but risks increased overhead

Interesting Sharing Issues

Things we care about when sharing resources

- Mix of small and large jobs makes sharing nodes valuable
  - *We would like to see maximum utilization of all node resources*
- Would like co-located jobs to not impinge on each others resources
  - *CPU*
  - *Memory*
  - *Disk space*
  - *I/O bandwidth*
- Ideally jobs should have their own security context
  - *No way to interfere with or communicate with each other*
  - *...Unless specifically requested*

We want strong isolation and efficient sharing, two opposing goals

Whole Node (or Larger) Allocations

Strong Isolation, But Low Granularity

- Supercomputing centers often allocate whole nodes or even require larger allocations
  - *The Texas Advanced Computing Center's Ranger cluster requires that users utilize full 16-core nodes*
  - *By far, the most popular approach today*
- Pros:
  - *Users can not interfere with each other's disk, memory, or network space and bandwidth*
  - *OS or hardware roblems triggered by short jobs do not effect long running ones*
  - *Security considerations are reduced due to lack of concurrent node access*
- Cons:
  - *Jobs must be of node granularity or resources are wasted*

Good for big science, but a bad fit for our job mix

Gang Scheduling

Time Sharing in the Large

- Time sharing on the scale of a whole or partial cluster
  - *Jobs are given a time slice (usually on the order of hours)*
  - *At the end of their time slice, the job is suspended and another scheduled or resumed.*
  - *Sometimes approximated with short maximum job run times.*
- Pros:
  - *Allows jobs to run without interference from each other*
  - *Partial results can be returned sooner than with run to completion*
- Cons:
  - *Context switch costs are high*
    - Network connection must be re-established
    - Data must be paged back in
  - *Lack of generally useful implementations*
  - *Less useful with small jobs, especially those that do not need full nodes*

Gang scheduling is useful, but not a good fit for our job mix

Single Application/Project (Sub-)Clusters

Maximum Isolation

- Clusters allocated on demand or for the duration of a project
- Systems like EmuLab, Sun's Project Hedeby, or the Cluster on Demand work at Duke allow rapid deployment
- Pros
  - *Complete isolation*
  - *Ability to tailor nodes to job needs*
- Cons
  - *Course granularity*
    - Does not easily support small jobs
  - *Expensive context switches (up to tens of minutes)*
  - *Users can interfere with themselves*
  - *No general way to recapture underutilized resources*

Powerful isolation, but high costs

Virtualization

A Cost Effective Route to Sub-Clusters?

- Allows relatively rapid deployment of node images
- Multiple images can share a node
- Pros
  - *Strong isolation*
  - *Ability to tailor node images to job needs*
  - *Possible to recovery underutilized resources*
- Cons
  - *Incomplete isolation due to shared hardware*
  - *Users can interfere with themselves*
  - *No way to efficiently isolate small jobs*
  - *Significant overhead*
    - CPU slowdown
    - Duplicate disk and memory use

Virtualization may make sub-clusters practical

Virtual Private Servers

An Alternative from the Internet

- Developed by the internet hosting industry to support large number of clients on a single host
- Pros
  - *Small overhead vs. virtualization*
    - Makes per-job images practical for small jobs
  - *Ability to tailor images to job needs*
  - *Only virtualize what needs virtualizing*
- Cons
  - *Incomplete isolation*
  - *Reduced flexibility in images vs. virtualization*
    - e.g. no Windows images on FreeBSD

A lightweight alternative to virtualization

Resource Limits and Partitions

Leveraging Existing Features

- All Unix-like operating systems support per-process resource limits
  - *Schedulers support the most common ones*
- Most support various forms of resource partitioning
  - *Memory disks*
  - *Quotas*
  - *CPU affinity*
- Pros
  - *Use existing operating system features*
  - *Easy integration in existing schedulers*
- Cons
  - *Incomplete isolation*
  - *No unified framework in most operating systems*
    - Irix per-job resouces and Solaris project are exceptions
  - *Typically no or limited limits on bandwidth*

There is room to enhance schedulers to use more OS features

Our Experiments

What Will Work on Fellowship

- We need a solution that handles our wide range of job types
  - *Single application/project clusters*
    - Fully isolate users
    - Require virtualization to be efficient in our environment
    - Don't handle very small jobs well
  - *Resource limits and partitions*
    - Implementable with existing functionality
    - Achieve useful isolation
  - *Virtual Private Servers*
    - Allow per-job differences in operating environment
    - Isolate users from changes in the kernel
    - Provide strong isolation for security purposes
- Resource partitions and VPS technologies will have similar implantation requirements

Focus on partitioning, then VPS technologies

SGE Shepherd Wrapper

Restricting Job Execution Environment

- The SGE shepherd is the parent of all processes in each job
  - *Collects usage statistics*
  - *Forwards signals to children*
  - *Starts remote job components (in tightly integrated jobs)*
- Original plan involved modifying shepherd to implement restrictions
- SGE allows specification of an alternate location for the `sge_shepherd` program
- We have implemented a wrapper script that runs the shepherd indirectly
  - *precmd hook performs setup*
  - *cmdwrapper hook adds additional programs to the front of the command*
    - i.e. `env FOO=BAR sge_shepherd`
  - *postcmd hook performs clean up*
  - *Implemented in Ruby*

Shepherd Wrapper allows rapid prototyping and implementation

Memory Backed Temporary Directories

Reducing Contention for Temporary Storage

- SGE manage paths for per-job temporary storage
  - *Creates a temporary directory on each node for use by each job*
  - *Points `TMPDIR` environmental variable to directory*
    - Well designed Unix programs store temporary files in `TMPDIR` by default
  - *After execution temporary directory is destroyed*
  - *These paths share a common parent directory*
    - Jobs that use too much storage can cause problems for others
- We have implemented a wrapper that mounts a memory backed file system (a swap backed md(4) device) over the SGE `TMPDIR`
  - *Users can request an allocation of a specific size*
  - *Since allocations are set at job start up time, jobs should not unexpectedly run out of space*
- As a bonus, memory backed storage will improve performance

Separating temporary storage improves reliability and performance

Memory Backed Temporary Directories (cont.)

Example

```
$ cat foo.sh
#!/bin/sh
echo "TMPDIR = $TMPDIR"
df -h ${TMPDIR}
$ qsub -l tmpspace=100m -sync y foo.sh
Your job 156 ("foo.sh") has been submitted
Job 156 exited with exit code 0.
$ cat foo.sh.o156
TMPDIR = /tmp/156.1.all.q
Filesystem    Size    Used   Avail Capacity  Mounted on
/dev/md0      104M    4.0K     95M     0%    /tmp/156.1.all.q
$
```

Quick and effective isolation of TMPDIR

Variant Symbolic Links

Why TMPDIR Is Not Enough

- Memory backed `TMPDIR` works for well designed applications
- Badly designed applications hard code `/tmp` which defeats `TMPDIR`
  - *Can result in exhaustion of shared resources*
  - *If full paths are hard coded can result in data corruption and bizarre failures*
    - Accidental sharing of data between jobs
    - Confusion in interprocess communications
- What we need is per-job `/tmp`
- Variant symlinks can provide this and other partial file system virtualizations

Memory backed `TMPDIR` only solves part of the problem

Variant Symbolic Links

Introduction to Variant Symlinks

- Variant symbolic links are symlinks that contain variables that are expanded at runtime
  - *Allows paths to differ on a per-process basis*
  - *Example*

```
$ echo aaa > aaa
$ echo bbb > bbb
$ ln -s %{FOO:aaa} foo
$ cat foo
aaa
$ varsym FOO=bbb cat foo
bbb
$ sudo varsym -s FOO=bbb
$ cat foo
bbb
```

Variant symbolic links provide partial file system virtualization

Variant Symbolic Links

Our Implementation

- Derived from DragonFlyBSD implementation
  - *Changed significantly*
- Scopes
  - *System > privileged per-process > user per-process*
  - *No user or jail scope (jail coming eventually)*
  - *Scope precedence reversed relative to DragonFlyBSD*
- Default value support
  - `%{VARIABLE:default-value}`
- Use to `%` instead of $ to avoid confusion with environmental variables
  - *Not using @ to avoid conflicts with AFS and NetBSD implementation*
- `/etc/login.conf` support
- No automatic variables (i.e. @sys)
- No setting of other processes variables

Focus on simple, easy to reason about primitives

CPU Set Allocator

Giving Jobs Their Own CPUs

- In a typical SGE configuration, each node has a “slot” for each CPU
- Jobs are allocated one or more slots
  - *One for plain jobs*
  - *One or more for jobs in parallel environments*
- No actual connection between slots and CPUs
  - *Badly behaved jobs may use more CPUs than they are allocated*
  - *Earlier versions of SGE supported tying slots to CPUs on Irix*
- We have used our SGE shepherd wrapper and the cpuset functionality introduced in FreeBSD 7.1 to bind jobs to CPUs
  - *Allocations stored in `/var/run/sge_cpuset`*
  - *Naïve recursive allocation algorithm*
    - No cache awareness
    - Try best fit, then minimize new fragments
  - *Should port easily to other OSes*

Tying jobs to CPUs keeps interference to a minimum

CPU Set Allocator Benchmarks

Benchmark Platform

- System
  - *Dual Intel Xeon E5430 @ 2.66GHz*
    - 8 cores total
  - *16GB RAM*
  - *FreeBSD 7.1-PRERELEASE (r182969) amd64*
    - Needed for `cpuset(1)`
  - *SGE 6.2*
- Benchmark
  - *N-Queens problem*
    - Simple integer workload
    - Minimal memory and no disk use
  - *`nqueens-1.0` (`ports/benchmarks/nqueens`)*
  - *Measured command: `qn24b_base 18`*
  - *Load command: `qn24b_base 20`*
    - Invoked as needed to generate desired load

Keeping the benchmark simple allows for easy reproduction

CPU Set Allocator Benchmarks (cont.)

Results

	Baseline	7 Load Procs	8 Load Procs	7 Load Procs w/ cpuset	8 Load Procs w/ cpuset
Runs	8	8	17	11	12
Average Run Time	345.73	347.32	393.35	346.63	346.74
Standard Deviation	0.21	0.64	14.6	0.05	0.04
Difference From Baseline		0.59	46.63	*	*
Margin of Error		0.51	10.81	*	*
Percent Difference From Baseline		0.17%	13.45%	*	*

*** No difference at 95% confidence**

CPU Sets improve predictability and performance

Conclusions and Future Work

The Future of Job Isolation

- Useful proof of concept isolations implemented
- Virtual private servers per job
  - *Isolate users from kernel upgrades*
    - Allow performance improvements without upgrade costs
  - *Allow multiple OS versions*
    - amd64 and i386 on the same machine
    - Full Linux environment on FreeBSD hosts
      - *DTrace on Linux*
- Limits on or reservations for network or disk bandwidth
  - *Network bandwidth limits possible for socket IO, hard for NFS traffic*
  - *Disk IO reservations a la Irix XFS could help some job type*
- Per job resource limits a la Irix jid_t or Solaris projects in FreeBSD

Job isolation is feasible and useful

Questions?

http://people.freebsd.org/~brooks/pubs/bsdcan2009/

Disclaimer

- All trademarks, service marks, and trade names are the property of their respective owners.