Using cpusets on a 256 CPU Origin 3800

Jim Glidewell & Barry Sharp

Boeing Shared Services Group

james.glidewell@boeing.com

The blessing and curse of ccNUMA

- Large memory available for single or multi-CPU processes
- Simple, very efficient MPI data sharing
- Relatively low cost compared to similarly sized “flat” memory systems
- Memory access speed depends on “nearby” memory
- The larger the system, the more critical data locality becomes
- It is essential to not oversubscribe CPU resources, or data locality may be “permanently” lost
- Very hard to detect when performance is degraded due to oversubscription and/or memory locality
    - Increased memory latency is “hidden” in increased CPU time
    - Router traffic gives some indication, but varies by application mix

The cpuset solution

- Introduced at IRIX 6.5.8
- Allows dynamic software partitioning of the machine
- Keeps data and processors co-located, eliminates interference from other jobs on the system
- Originally tied to Miser, which had rather mixed reviews
- Appeared to be little used in the field, with a few large & notable exceptions (NAS)
- Seemed to add undesirable complexity and rigidity
- Didn’t seem worth the effort on our 64 CPU Origin 3800

Factors which drove us to examine cpusets

- Recognition that our upgrade to 256 CPUs would make memory locality issues more critical
- Huge variations in run times for one of our “workhorse” aero applications (Overflow)
  - CPU time on an identical case varied between 90,000-200,000 seconds
  - Even larger variation seen by end users on similar cases
- Reports of significant performance improvements on Overflow at NAS using cpusets
- The choice of PBSPro as our batch scheduling system
- Encouragement by onsite SGI analysts with reasonable comfort and familiarity with cpuset usage

Our early experiments with cpusets

- Overflow test case - consumed between 90,000 and 200,000 CPU seconds
- Initial testing simply created a cpuset "on the fly" and assigned process to it
  - Little improvement over non-cpuset runs - 87,000 - 186,000 CPU seconds
  - Suspected interference with existing processes resident in the designated P-bricks
- Chose 87,000 seconds as a goal for consistent run time
- Consistent times were only achievable when we allowed the cpuset to "dry out" by defining the cpuset and then leaving it idle for up to 24 hours prior to executing the job
- Given this experience, we concluded that the only way to ensure good performance was to restrict **all** processes to a cpuset

Interim solutions

- Given the rather strong evidence that our failure to adopt cpusets was having a severe impact on one of our key customers, we felt we needed to offer *something* in this area
- Did not want to use PBSPro managed cpusets, as PBSPro had the unfortunate behavior of creating a 4-CPU (one node) cpuset for a single-CPU job
- Luckily, we had just increased our capacity by 4X (64 -> 256 CPUs), which gave us some room to experiment

Interim solution #1

- Interim solution #1a
  - Establish a fixed 64 CPU cpuset (overa)
  - Let the Overflow users manage this cpuset among themselves
  - Users manually assigned their main program to the cpuset within the PBS batch job
  - This didn’t work all that well...
- Interim solution #1b
  - Provide a script which checked whether the “overa”cpuset was idle
    - “idlecpuset” returns the name of an idle cpuset (if available) and null otherwise
    - User PBS scripts could then test for an available cpuset, and either assign their main program to a cpuset, run outside a cpuset, or exit
    - This allowed for the possibility of multiple cpusets, and allowed users to make use of cpusets without coordinating among themselves

Interim solution #2

- Multiple fixed cpusets, created on demand
- Modified “idlecpuset” script to either:
  - Return the name of an idle cpuset (if one was available)
  - Issue a request that a cpuset be created (via a flag file), then wait and retest for an idle cpuset in a few minutes
- Separate “createcpuset” daemon
  - Monitors cpuset status
  - Creates cpuset upon request
  - Deletes idle cpusets
- This solution actually worked fairly well, although:
  - It restricted the users to a single cpuset size (64 CPUs)
  - Cpusets were being created “on top of” existing processes in many cases, resulting in less that optimal performance

The “Flythru” cpuset

- Flythru is a Boeing developed system for sharing Catia geometry data with Unix-based workstations
- We were asked to host Flythru on our Origin 3800
- The two major functions of a Flythru server are file sharing (using NFS) and “Update” which synchronizes the local data models with the reference Catia database
- Our focus on cpusets **had** been protecting a single application from the “noise” of the other processes on the system
- We discovered fairly late in the migration that Update generated anywhere from 10K to several 100K processes per hour!
- This process count has a fairly disastrous effect on other processes, “stirring the pot” and effectively randomizing process placement on CPUs
- We immediately created a single-node (4 CPU) cpuset to hold the Update processes, which has mitigated the issue (for the most part…)

Moving cpusets into the mainstream - PBSPro 5.2

- We had deferred the management of cpusets by PBSPro until a solution to the “single CPU job” dilemma was available
- Veridian offered a solution to that issue with PBSPro 5.2 - “shared cpusets”
- With this feature, “small” jobs will be placed in a shared cpuset with as many as 3 other jobs (on a 3800, depends on job’s memory and cpu requirement)
- This allows a single system to run a mix of single-cpu and parallel jobs without significant concern about idle or wasted resources
- Based on continuing needs of our Overflow users for more flexible cpusets, and with the expectation that the transition to PBS-managed cpusets would be essentially transparent to our other users, we decided to move forward with PBSPro 5.2 and cpusets…

Our initial configuration

- A 12 CPU, 18GB boot cpuset (handles all processes except those specifically assigned to another cpuset)
- A 4 CPU, 6GB "flythru" cpuset (to isolate the Flythru update application)
- Remaining 236 CPUs & 354GB memory scheduled by PBSPro using cpusets
- PBSPro 5.2, shared cpusets enabled, one local scheduling mod, configuration defaults used whenever possible
- IRIX version 6.5.12f

A Rough Start…

- After a moderate amount of testing on our 4-CPU test partition, and further limited testing on the 252 CPU partition, we installed PBS Pro 5.2 with cpusets enabled
- System was heavily loaded
- We encountered a number of problems initially
- Many of the problems were greatly aggravated by a difficult-to-diagnose hardware problem which caused multiple system outages over a week’s time

A Rough Start (continued)

- Since the installation, we have seen a number of issues, including:
  - Failure to schedule more than 1/2 of the machine (workaround - schedule by ssinodes, rather than mem & cpu)
  - Job aborts on shared nodes, due to “out of memory” conditions (workaround - change the cpuset creation flags from “POLICY_KILL” to “POLICY_PAGE”)
  - Jobs stuck in “E” state (workaround - kill server with a “qterm -t quick” and restart server daemon)
  - Failure to properly recover shared cpusets after restart of system or pbs_mom
  - User concerns about reduced overall system throughput
  - Excessive CPU consumption by pbs_mom
  - Increased swapping
- To their credit, the PBSPro support staff have been extremely responsive and helpful in trying to get us through this period

The need for visibility

- Prior to the “shared cpuset” feature, there was essentially a 1-to-1 correspondence between PBS jobs and cpusets - but this is no longer the case
- Given that is no longer the case, we really felt we needed a tool which would display the relationships between PBS jobs and cpusets, hence “cpurep”was created
- “cpurep” provides 3 blocks of data: cpusets, PBS jobs, and overall subscription
- This tool has been very valuable to us in monitoring and debugging PBS problems, and in understanding the “shared cpuset” scheduling methodology
- In addition, we have created a Performance Co-Pilot tool which allows up to view the CPU and memory usage in a cpuset - “pmgcpuset”
- We are exploring other methods of tracking and displaying information about cpusets and their resource utilization

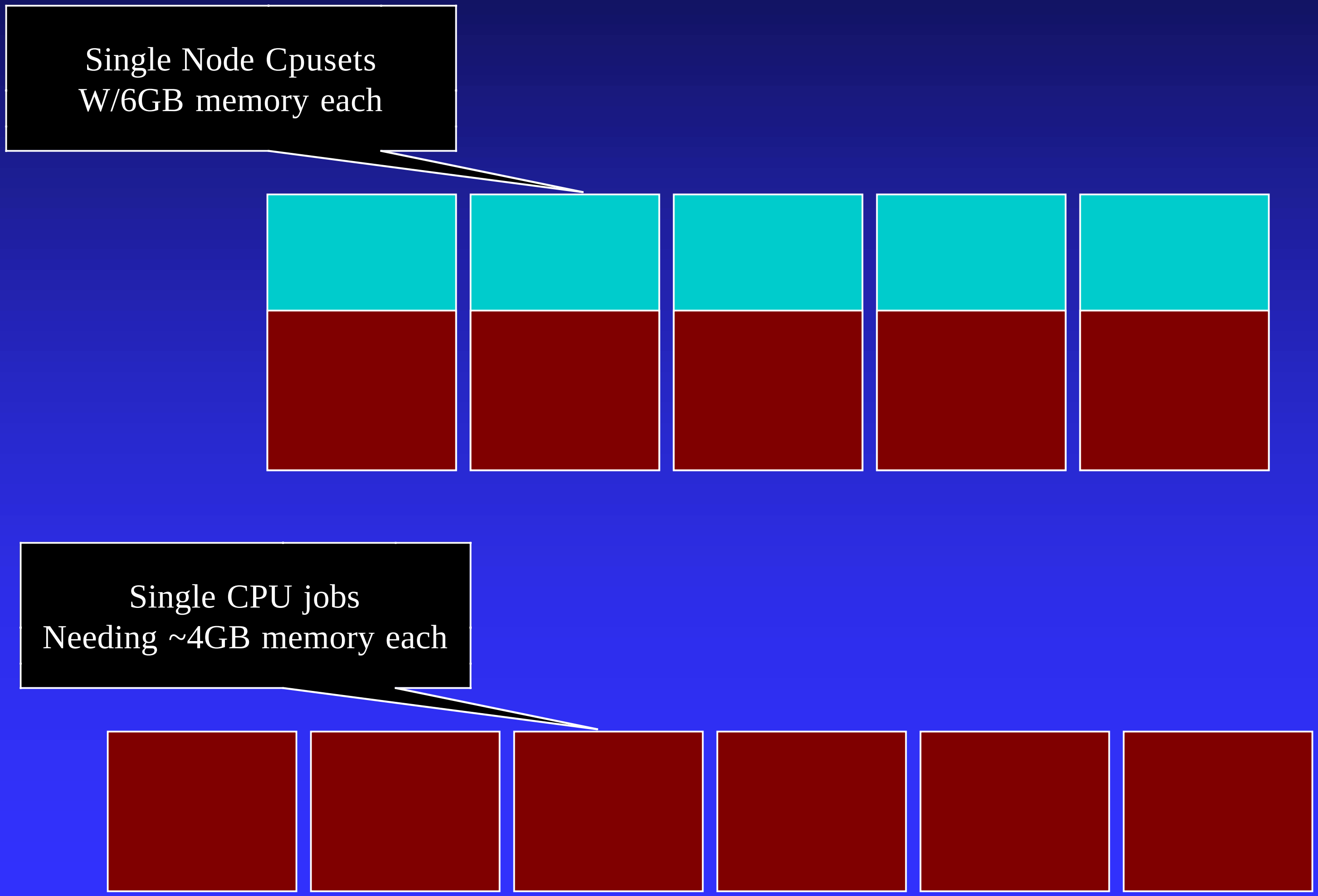
Cpurep output

```
origin 11% cpurep

cpuset        cpus    procs   PBS_jobs
------        ----    -----   --------
boot          12      377
flythru       4       56
41013.or      64      66      41013(xxx7307)
40497.or      4       7       40523(yyy2754)
40550.or      4       12      40576(xxx7307) 41014(tecxxx1) 41015(tecxxx1) 41016(tecxxx1)

Job_id   user     Irix_jid             cpuset     cpus  shared?
------   ----     --------             ------     ----  -------
40523    yyy2754  0x2a1900000001120b   40497.or      4
40576    xxx7307  0x2a1900000001b712   40550.or      4     *
41013    xxx7307  0x2a19000000036cfe   41013.or     64
41014    tecxxx1  0x2a19000000036d53   40550.or      4     *
41015    tecxxx1  0x2a19000000036d54   40550.or      4     *
41016    tecxxx1  0x2a19000000036d55   40550.or      4     *

Assigned Nodes Map:

XXXX........X........XXXXXXXXXXXXXXXXX........................

Total CPUs allocated = 88. Nodes allocated = 22.

Total CPUs free = 164. Nodes free = 41.
```

The “Packing” Problem

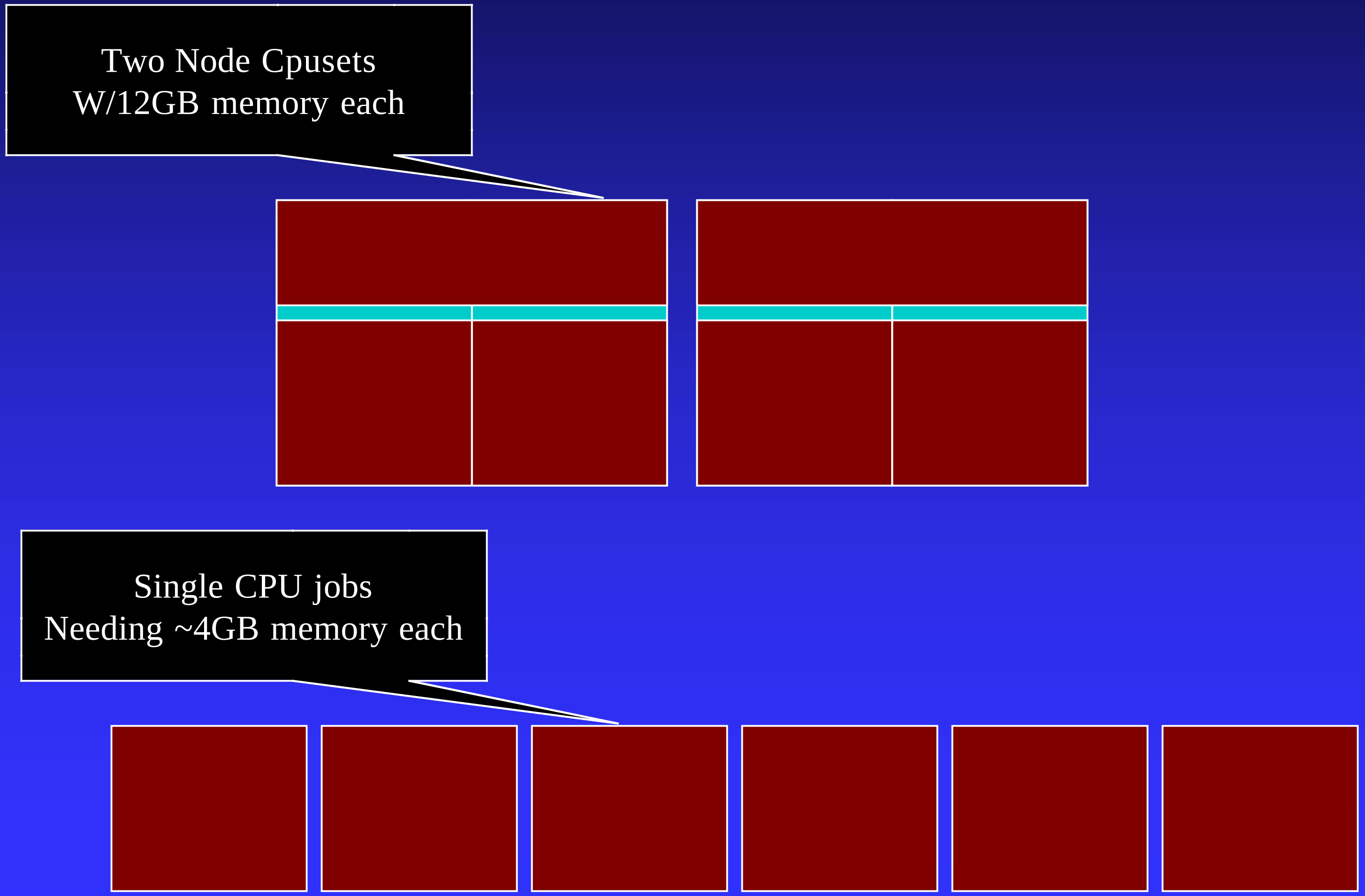
- When PBSPro encounters a “small” job which will not fit in an existing cpuset, it creates a new single-node cpuset
- This can be an issue if there are a large number of jobs which do not pack neatly into single nodes
- This issue has already arisen at our site, and has raised concerns about reduced throughput

The “Packing” problem



One possible solution…

- Allow sites to define a “minimum allocation size” for shared cpusets



User feedback

- Our Overflow customers are *very* happy with our implementation
- A few users saw initial problems, which we believe have all been resolved
- Most users apparently didn’t notice the change - which was our intention

Cpusets - worth the trouble?

- For us, cpusets have provided a solution to a couple of vexing issues:
  - Poor performance on large parallel jobs
  - Badly behaved applications causing overall adverse system impacts
- Despite the "teething pains" we have experienced, we still believe that cpusets are the best solution we have to offer our users, and that the PBSPro shared cpuset feature will make a significant difference in our ability to get the maximum efficient usage of the Origin 3800
- For sites that do not have a batch system which offers cpuset support, creating special-purpose cpusets is a reasonable option for certain situations such as:
  - Problematic applications (high process counts)
  - Large parallel applications that run poorly in a normal batch environment
  - Other special situations… ???

Future areas of effort

- Better cpuset visibility tools
- Better support for cpusets under Performance Co-Pilot (PCP)
- Investigation into why other sites still report variability when using cpusets
- Smooth out the rough edges of shared cpusets and PBSPro 5.2
- Further discussions with Veridian on solutions to the packing problem