

Introduction

Within Linux, the process scheduler is responsible for distributing the work load between multiple CPUs on a multiprocessor system. By using Processor Affinity settings, one can manually override the scheduling algorithm and specify a particular CPU for a particular task. This can, in certain circumstances, increase the efficiency of a task by running it on the same processor - which may have cached it locally, or may be closer to a piece of needed hardware.

Non-Uniform Memory Access (NUMA) is an architecture found in some multiprocessor systems where locally accessed memory will be faster than accessing memory in a neighboring location.

Objective

Our project involved modifying a high performance computing cluster using processor affinity settings and comparing several benchmarking metrics with baseline results to see where improvements could be made in a production environment.

Testbed

Our team built a 10 node cluster using the latest versions of CentOS Linux and the Perceus cluster provisioning suite and interconnected with SDR InfiniBand and Gigabit Ethernet. The hardware on each of the nodes (two of which were former Roadrunner Phase I nodes) was slightly different from the others, giving our group an excellent vehicle to test the effects of changes in a variety of hardware situations.

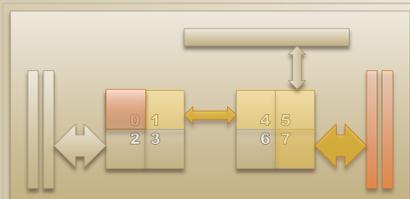
Methods

Linux kernel objects called CPUSets allow a system administrator to partition sets of processors and memory into specific execution areas. This forces the sequestration of processes to specific hardware. These rules are inherited to children processes, which allows secure, 'jailed' subdivisions of hardware in high performance computing situations. CPUSets are, however, limited to already running tasks.

The NUMACTL tool, however, can be used to implement memory policies that compliment an already invoked CPUSet arrangement by affecting as a process is invoked. Hardware visualizations and statistics are also part of the tool package, and make the administrator's life easier.

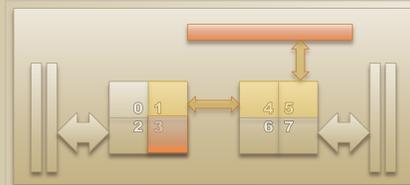
Hardware Interrupt Requests (IRQs) can flood a system making it unstable; having the ability to cloister IRQs to a specific processor can clean up system noise

and prevent unnecessary communication between cores and sockets. Linux includes features to set IRQ affinities via the file system, in much the same way as CPUSets.

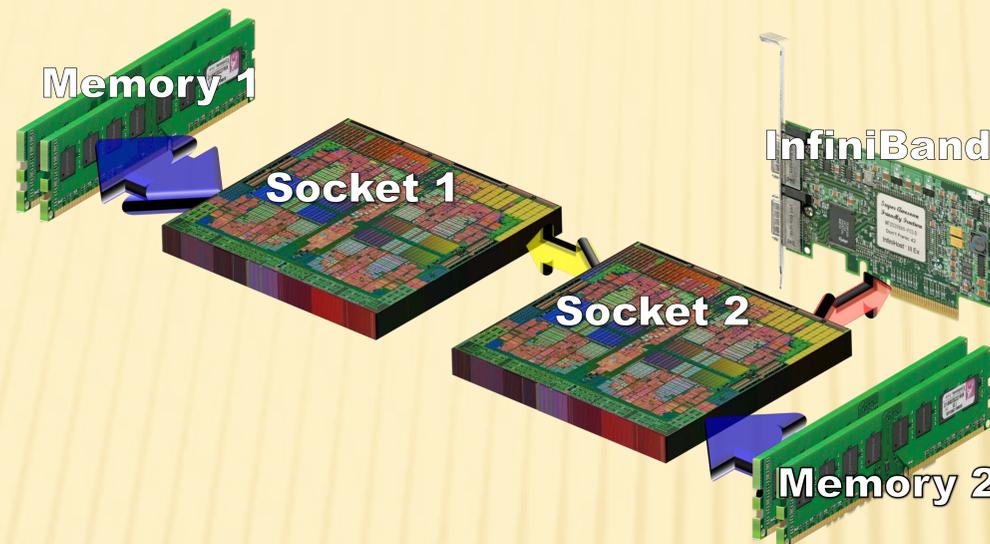


A processor accessing non-local memory will have longer access times than if it accessed its local memory.

In order to speed and ease testing, our group developed a script that implements a CPUSet and IRQ affinity policy from flat configuration files. This was a great way to implement an affinity policy, and may be useful to cluster administrators.

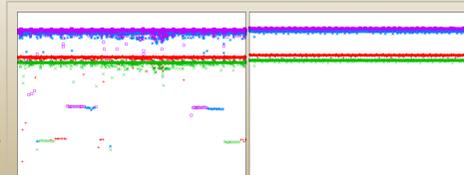


Interrupts from hardware, going to non-local processors, can cause variances in user application performance.



Operating System Noise

In an HPC cluster, the operating system (running on each node) can cause system noise through 'cache thrashing' and IRQs bothering an otherwise healthy user application. Caging the system processes to an individual CPU, can prevent shared caches across the system from being repeatedly dirtied, thus speeding up memory operations that depend upon the cache.



These graphs show memory bandwidth tests on a system with the default scheduling configuration versus one that has been optimized for a reduction in system noise using CPUSets.

Benchmarking

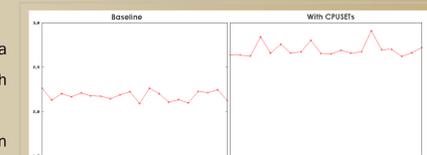
We concentrated on three system bottle-necks for our benchmarking to see what effect changes in affinity had on system performance: memory bandwidth, IP socket bandwidth and InfiniBand RDMA performance.

For our memory bandwidth tests, we used a modified version of the Stream benchmark which allowed for more detailed logging.

We used IPerf to test both TCP and UDP speeds on our Gigabit Ethernet and Single Data Rate (SDR) InfiniBand.

Remote Direct Memory Access over InfiniBand is a big deal in cluster computing. Benchmarking this against a baseline was an important aspect of our project. QPerf, included in OFED, was used for our measurements of both unidirectional and bidirectional bandwidths and latencies.

Modifications to the system IRQ affinities also found network performance increases under certain circumstances. On one node we were seeing increases of over 30 Mb/sec using Gigabit Ethernet.

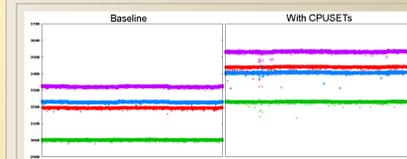


Network variances seen on our two socket nodes before CPUSets smoothed things out on the right.

Various CPU and IRQ affinities in our tests using QPerf to measure RDMA bandwidth over InfiniBand yielded no noticeable changes. However we are confident that additional testing is required using other architectures that were not available to us during our project.

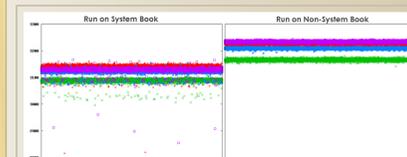
Results

Our tests included one data point within the cache and one giving a memory bandwidth only test. Speed results on each of our testbed systems varied. From increases to decreases.

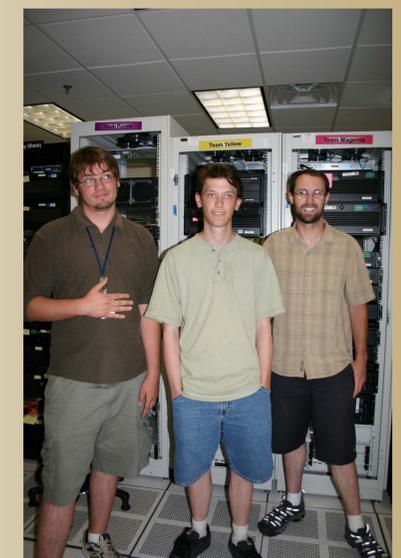


An increase in non-cache memory speeds can be seen with simply setting the system processes to a single processor.

In most situations, however, we saw speed increases and noise reduction, as seen in the graphs below.



On our Road Runner Phase I node, we were able to see significant noise reduction and speed increases when we switched the memory bandwidth test to non-system cores.



Conclusions

Throughout our research, we found highly differential results that depended on the hardware being used. The key advantage to using affinity is that system noise can be reduced. We've seen both average bandwidth increase and decrease while setting affinity, the commonality being reduced noise. The disadvantage is, many times this can come with a cost of performance. In some models affinity is used to isolate system process to one CPU while user processes can use the rest. In our case, this meant that 1/8 of our systems processor power was not usable by user applications. This becomes a more

manageable number if you increase the number of processing cores available for system usage. The downside to isolating one core for system processes is that typically the system will never fully utilize one core, so in essence your throwing away some CPU cycles. What you have to decide on a system by system basis is if losing CPU cycles is worth having greatly reduced noise. In recent years the Linux kernel has been updated to support CPUSets. In the future, Linux is going to have to revamp the way it schedules processes as more complex systems become the norm. In the interim it is necessary to use middle ware to optimize system performance. To fulfill one of the project goals, a system was developed that automates the process of creating an affinity policy on large clusters.

Future Work

It is important to realize that hardware changes rapidly. Different hardware requires different strategies need to be taken as far as affinity scheduling goes. When CPU's begin being produced with more cores on a single die, it becomes an option to jail system processes to one core to reduce OS noise and leave the remainder for the user processes. While this isn't practical in a four core system, it certainly becomes practical in a twenty core system. Research in affinity should continue as computer hardware advances. Each hardware configuration must be analyzed separately in order to determine the appropriate

operating system configuration for that hardware. In the future it will be necessary to automate the systems analysis to determine the best affinity configuration for a given system. Due to time constraints, our group was unable to fully look into the affinity scheduling capabilities of Torque and other job schedulers.