Optimizing Shared Resource Contention in HPC Clusters

Sergey Blagodurov
PhD Candidate
Simon Fraser University
sergey_blagodurov@sfu.ca

Alexandra Fedorova
Associate Professor
Simon Fraser University
fedorova@cs.sfu.ca

I. Introduction. Contention for shared resources in High Performance Computing (HPC) clusters occurs when jobs are concurrently executing on the same multicore node (there is a contention for allocated CPU time, shared caches, memory bus, memory controllers, etc). This contention incurs severe degradation to workload performance and stability and hence must be addressed. The state-of-the-art HPC clusters, however, are not contention-aware, with no virtualization supported to mitigate the contention effects through live job migration. The goal of this work is the design, implementation and evaluation of a virtualized HPC scheduling framework that is contention aware.

II. Proposed solution. We introduce Clavis-HPC, a novel contention-aware virtualized HPC framework. Here is how it solves the contention problem:

1) We monitor job processes (ranks) on-the-fly and classify them into two categories with respect to their shared resource usage: (a) process is a *devil* if it is memory intensive, otherwise it is a *turtle*. From our previous work, we know that devils compete for shared resources while turtles do not [1, 2, 4]. We classify the ranks into a particular category on-the-fly by using hardware performance counters (available in all modern processor models), and a robust machine learning technique which we previously introduced in [3].

2) We develop a multi-objective algorithm called Clavis-Cluster to periodically find a new job schedule according to 3 competing goals: (a) minimize the number of devils on each node; (b) maximize the number of communicating processes on each node; (c) minimize the total power consumption of the cluster. We achieve this through minimizing a weighted sum of the three goals with Choco solver.

3) After the new cluster-wide schedule is found, we enforce it by introducing a low-overhead live migration into the cluster: the cluster job scheduler (Maui) places processes into OpenVZ virtual containers (VMs), while Clavis-Cluster migrates containers between the nodes. Clavis-Cluster decisions are completely transparent to Maui, which only schedules workload within containers.

Please refer to the poster draft to see how framework works step-by-step.

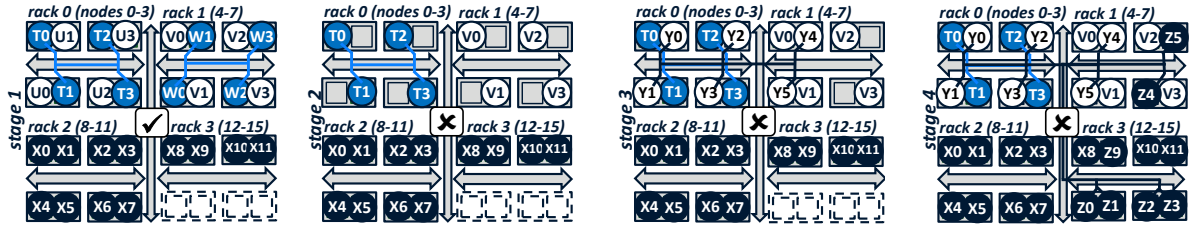
III. Experimental setup. We demonstrate the benefits Clavis-HPC provides on an **India** cluster hosted by FutureGrid. The nodes have the following hardware configuration: IBM iDataPlex (Intel Xeon X5570 Nehalem) servers have 8 cores placed on two chips. Each chip has an 8 MB L3 cache shared by its four cores. Each core also has a private unified L2 cache and private L1 instruction and data caches. It is a NUMA system: each CPU has an associated 12 GB memory block, for a total of 24 GB main memory. The servers were configured with a single SCSI hard drive. The nodes are connected through 1GbE and InfiniBand networks. We conduct experiments on 128 cores (16 node) cluster that we create within the India infrastructure. In our setup, nodes are divided among 4 racks that are connected via a slow link. We use the *netem* kernel module to induce latency between the racks, thus emulating a geographically distributed system. We use Maui as our job scheduler, Torque as a resource manager, SPEC MPI2007 and SPEC HEP as a cluster workload. All nodes were running Linux 2.6.32.

We create a job queue inside the cluster. Jobs are scheduled to run in the order of their arrival. Some jobs have dependencies: they cannot run before the job they depend on completes. In the figures that follow we will show the placement of jobs in the cluster across several stages. A stage begins when a new set of jobs is assigned to run and ends when those jobs complete.

We compare *Clavis-HPC* results with that of the *Expert* scheduler. The *Expert* job placement is performed by a knowledgeable HPC administrator who can peek one step ahead into the job queue and who is willing to manually place jobs to nodes based on the knowledge of their performance characteristics (contention-intensive or not, communication-intensive or not). Since typical HPC clusters do not use virtualization, we assume that the administrator is unable to migrate jobs after they began execution. Nevertheless, clever placement of jobs prior to beginning of execution gives an important advantage, so we expect to see very good performance and energy with the *Expert* scheduler. At the same time, it is not feasible to expect job scheduling to be performed manually by a human, so *Expert* is an idealized, unrealistic scenario.

On India cluster, servers consume a significant amount of power when they are idle, often more than 50% of peak power. To show how *Clavis-HPC* adapts its decisions to changes in hardware, we here assume that the administrator enabled dynamic power management algorithms on the nodes in order to reduce their idle power. We assume that as a result the idle power consumption on India has been reduced from 50% of the peak power, to only 25%.

IV. Results. The job queue in these runs is comprised of 7 jobs, T through Z, three of which (U, V and X) are SPEC HEP jobs. SPEC HEP is the official CPU performance benchmark used on clusters processing LHC data. It is very common for LHC workloads to utilize huge parts of the cluster, so the biggest job X (soplex) in our testbed requires 37% of the computing facilities (12 containers). The sheer size of this job makes it difficult to improve its perfor-



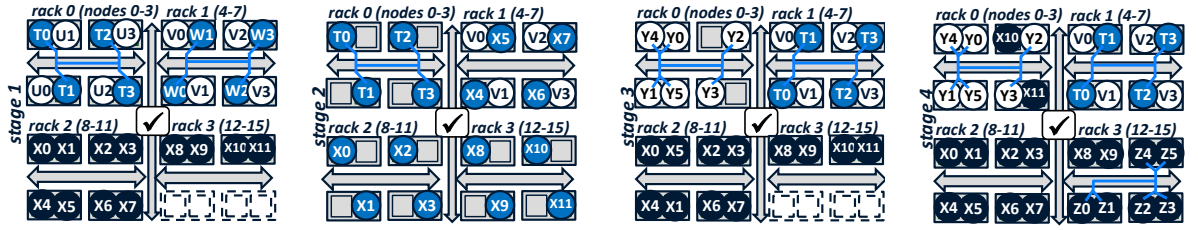
(a) Expert manually shuffles containers of T, U and V, W to boost T and W.

(b) Expert is unable to spread containers of X apart to increase performance.

(c) The job Y now has to suffer remote communication.

(d) Expert is unable to reshuffle containers of X and Z to prevent remote access for Z.

Figure 1: Expert on India cluster, requires offline profiling and regular human involvement.



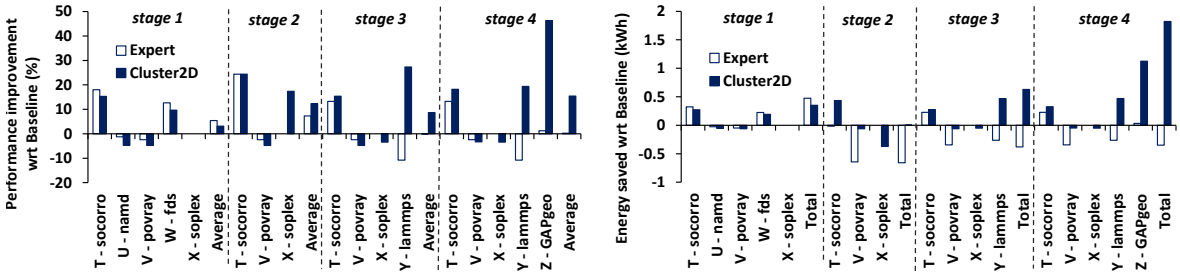
(a) *Clavis-HPC* dynamically collocates T, U and V, W.

(b) *Clavis-HPC* increases X's performance before reservations of Y and Z.

(c) *Clavis-HPC* puts Y within a rack to prevent slower remote communication.

(d) *Clavis-HPC* reshuffles containers of X and Z to avoid remote traffic for Z.

Figure 2: *Clavis-HPC* automatic experiments on India (initial placement is omitted due to space limitations).



(a) The performance results of India cluster runs.

(b) The energy measurements from India cluster runs.

Figure 3: Experimental results on India cluster.

mance in stage 1. The devil jobs T and W, on the other hand, can receive more than 10% boost if we pair them with turtles U and V. Both *Expert* and *Clavis-HPC* do just that (Figures 1(a) and 2(a)).

Despite our cluster being nearly fully utilized most of the time, in stage 2 it happens to have a pocket of idle resources, equivalent to six idle nodes in total (Figure 1(b)). This is because jobs Y and Z that should run next have advanced reservations and are waiting for their time slot. This opportunity is seized by *Clavis-HPC* to spread the containers of X on different nodes (Figure 2(b)), giving X a 20% boost in performance (Figure 3). By letting X use more nodes, *Clavis-HPC* sacrifices energy. The loss in energy is rather small due to using dynamic power management, and *Clavis-HPC* is aware of that, because we calibrated its weights using the power data from the "energy-savvy" India cluster. If a similar situation had occurred on a cluster, where idle power consumption is much larger, *Clavis-HPC* would have consolidated X, because power loss outweighs performance benefits in that case.

The jobs Y and Z on stages 3 and 4 arrive at the time when every rack on the cluster is partially loaded. As a result, *Expert* is not able to place them entirely within a rack, triggering slow cross-rack communication. Only *Clavis-HPC* is able to avoid this by reshuffling containers of already running jobs, thus freeing some rack space for the newcomers. Because of that, *Clavis-HPC* outperforms *Expert* in both performance and energy savings (Figure 3).

1. REFERENCES

[1] S. Blagodurov, S. Zhuravlev, M. Dashti, and A. Fedorova. A Case for NUMA-Aware Contention Management on Multicore Systems. In *USENIX ATC*, 2011.

[2] S. Blagodurov, S. Zhuravlev, and A. Fedorova. Contention-aware scheduling on multicore systems. *ACM Trans. Comput. Syst.*, 28:8:1–8:45, December 2010.

[3] T. Dwyer, A. Fedorova, S. Blagodurov, M. Roth, F. Gaud, and J. Pei. A practical method for estimating performance degradation on multicore processors, and its application to HPC workloads. In *SC*, 2012.

[4] S. Zhuravlev, S. Blagodurov, and A. Fedorova. Addressing Contention on Multicore Processors via Scheduling. In *ASPLOS*, 2010.