Managing Atypical HPC Jobs
With Eden

Scott Simmerman, NICS
Here is a scenario that calls for a different parallel programming paradigm than the typical HPC MPI approach. We want to simply run a bunch of scripts with different input parameters—sometimes called a ‘parameter sweep’. This approach goes by several names: embarrassingly parallel, simply parallel, many-task computing, or course-grained parallelism. The main point being that there is no synchronization or communication between the separate, parallel instances.
At NICS, we encountered this need from users of Nautilus because of its unique architecture. Since it operates as a single system image with lots of cores and lots of memory, users could run code atypical to HPC such as Python, Matlab, and Java using the whole machine. So this usually involved running lots of serial code concurrently. This led to the need for efficient scheduling and management of such scenarios.
Existing Solutions?

- GNU parallel
- Parallel Command Processor (pcp)
- Hadoop
- Swift, Nimrod, Condor

We examined some existing solutions for dealing with this ‘many-task computing’ paradigm. GNU parallel is a very handy tool that is designed to work in the Unix pipeline approach. However, it uses ssh to launch jobs across many machines--something we can’t do in the HPC environment. PCP uses MPI to wrap around and launch multiple serial jobs. Hadoop is a very different animal--requiring your code to fit into the map–reduce paradigm and using Java. There are other, larger frameworks such as Swift, Nimrod, and Condor that work by distributing work across disparate computing resources. We desired something very simple and easy-to-use--no ssh, no MPI, no compiling or installing, no extra languages, no hassles.
So coming back to our problem. How do we go about running all of these scripts in parallel?
Naive approach #1. One could submit a separate batch job for each script. You can see what a headache this would be to manage. However, some users continue to try this approach, ‘stuffing the queue’ with hundreds of jobs and annoying other users, not to mention the sys admins.
Naive approach #2—-a little better but still limited. Here, the user submits one batch script which runs all of their commands in the background. One drawback to this approach is that it relies on the operating system to place the processes on computing cores, not guaranteeing one dedicated core per process. Another drawback is the lack of flexibility or scalability. What if you needed to run 1000 commands and there were only 32 cores available?
So we developed a simple framework called Eden. Eden provides a scalable, easy solution to managing multiple runs of serial code. The process to run Eden involves: (1) make a list of commands to run, (2) make a header for the batch script specifying number of cores needed, (3) put these two files in a ‘run directory’, and (4) run Eden.

Eden takes care of automatically generating and submitting a PBS batch script, running all of your commands efficiently using available resources, and collecting timing and output from all runs.
Eden is a master–worker framework—nothing new. The master process runs on the login node where you start the Eden job. It submits a batch job to the scheduler which starts up worker processes on the compute nodes. These worker processes communicate with the master via socket connections. The workers continue to execute commands from the master until your list is exhausted.
The batch job that is submitted looks similar to the one shown earlier. But instead of running your commands in the background, it is running the worker processes, one on each CPU requested.

It’s not shown in this example, but Eden uses the ‘dplace’ command on Nautilus to pin worker processes to CPUs, guaranteeing dedicated placement.
Here is a more detailed schematic of how Eden is put together. Eden uses mostly shell scripts with a little tcl for socket handling. The entry point eden.sh launches the adam.sh script which acts as the master process. Adam reads in commands, generates a batch script, submits the batch script to Torque using qsub, manages distribution of tasks to the worker processes (abel.sh), and keeps a log of completed processes in a ‘commands_done’ file. The eve.sh script takes care of post-processing, collecting all timing and output from the jobs.
Eden is installed on Nautilus and Kraken at NICS. This URL has detailed instructions and examples on using Eden.

http://www.nics.tennessee.edu/eden