MPI jobs on OSG

Mats Rynge <rynge@renci.org>
John McGee <mcgee@renci.org>
Leesa Brieger <leesa@renci.org>
Anirban Mandal <anirban@renci.org>

Renaissance Computing Institute (RENCI)

August 10, 2007

Introduction

The Engagement VO has been established to support the activities of recruiting new users from diverse scientific domains with a goal of demonstrating the efficacy of the OSG infrastructure beyond the physics community. During our interactions with potential users, one recurring question was whether OSG supports MPI jobs. As the Engagement VO has had great success getting new users running serial jobs, it was decided that we would explore the current state of support for parallel jobs on OSG resources, especially within the context of the Weather Research and Forecasting model (WRF).

The approach taken was to get MPI jobs running on OSG using any necessary workarounds, document the process, and catalyze a discussion within the OSG community on the areas in which the infrastructure could evolve to enable large scale production-class support for the WRF model in particular, and for MPI jobs more generally.

In the end, we were able to get MPI jobs running at a few sites. The sites where we successfully ran these jobs can be characterized as being in one of three categories: sites on which we had interactive login rights; sites where we could submit fork jobs which in turn used the local scheduler command line tools to submit the MPI job; and finally, sites that had (jobtype=mpi) correctly configured. This brief paper describes our experience with running these jobs and probing the OSG infrastructure for MPI support.

Job Environment

A major difficulty for MPI job submission to OSG is with the job environment. When a job is accepted via the Globus gatekeeper, the job parameters are passed through an interface to create a submit file for the local scheduler, and then the job is submitted to that local scheduler. The local scheduler usually sets up a standard environment for the job, but the OSG CE software stack replaces this environment to include a set of OSG specific environment variables, such as $OSG_APP and $OSG_DATA. The problem with this process is that the local scheduler's environment variables are not preserved. The local batch scheduler's environment is especially important for MPI jobs. For example, PBS sets the PBS_NODEFILE variable listing the compute nodes that are assigned to the job. A serial job can ignore this, as the only node assigned to the job is the node the executable is started on. For a parallel job, PBS_NODEFILE is required to know where to start up the peer processes. In the case of MPI, this information is used by mpirun and mpiexec.

MPI Implementations and Interconnects

The most confusing and time-consuming part of running the first MPI job through Globus on a resource was compiling and linking the application with the specific MPI implementation and version so the application would work with the startup application (mpiexec/mpirun) and the interconnect on the compute element.

MPI has many implementations (MPICH, OpenMPI, OpenMP, IBM, …). We created statically linked binaries for a couple of these implementations and tried to get them running on various sites. The implementations we built against were MPICH-1.2.6, MPICH2-1.0, LAM-7.1.3, and OpenMPI-1.0, all with Intel compilers. We also used OpenMP on Purdue-Caesar. Using precompiled WRF binaries has been difficult as it is not clear when the binary does not match up against the local MPI version. Most of the time, there are no errors, and the model simply does not run. We have been lucky on a couple of occasions when trying to match the build/environment. The best example is the Vienna Ab-initio Simulation Package (VASP), which we built locally. Our libraries just happened to match up with Caltech's, and the code ran using the Globus gatekeeper without any issues. As we knew matching MPI version would be a major task, we picked our test sites with that in mind. Some of the sites were selected for testing because we could easily find mpirun/mpiexec, and some sites were selected because we could get temporary interactive logins to probe and debug the system much better than is possible via the grid interfaces.

One issue with the many different MPI implementations is that it is difficult to probe the compute elements to find out what is available. Questions that need to be answered before running an MPI job include:

- Which implementations are available?
- Which implementation is the default?
- Does the implementation need booting? If so, how?
- Which tool is used to launch the MPI application - mpirun or mpiexec?
- Which interconnects are available?
- Any deploy/implementation-specific options?

Submitting a job with (jobtype=mpi) should hide most of these questions. Globus should use the default mpiexec or mpirun, and set up the necessary environment. The only site we have found so far with this correctly configured is Purdue. For sites with multiple implementations or interconnects, multiple Globus jobmanagers could be provided, one for each MPI job type.

The second option we tried was (jobtype=single)(count=n) and using mpiexec/mpirun as the executable, but we then ran into the environment problem mentioned above. We did not get this working at any site, but it should be possible with a correct environment.

The last thing we tried, and for which we had some limited success, was to submit a fork job, which would then submit a job directly to the local scheduler. We note however that this is not an acceptable long term solution, as it is likely to be against local site policies. However, with enough probing and trying many jobs with different options, we did succeed at getting simple MPI jobs to work via this method.

We believe that for OSG to provide resources for MPI jobs, sites must advertise MPI capabilities and pre-determined details about the MPI implementation and deployment. OSG would also need to investigate mechanisms to provide access to multiple implementations/interconnects installed at the same site.

For performance reasons, it could be argued that sites without a low latency interconnect (Myrinet, Infiniband, …) should advertise MPI capability with some explicit characterization of the interconnect capabilities and limitations. This will help ensure that only appropriate sites can be selected for applications that require a lot of inter-processor communication. For example, our WRF test case requires about 90 minutes to run on 192 nodes when using Infiniband, however, the same computation required 12 hours on the same 192 nodes with 1Gb Ethernet TCP/IP for inter-process communication. OpenMP sites are not affected by this degradation, but should advertise OpenMP as the interconnect so that the user understands how to build their code.

MPI Compilers / Interactive Logins

For OSG sites to become efficient and effective providers of MPI resources, we believe that access to locally installed MPI compilers and possibly even interactive logins should be considered, solely for the purpose of building the binaries.

Some of the sites we have talked with believe that the user should not have to compile anything on the target resource, and that prebuilt static binaries should be used. This is a reasonable approach for serial jobs and is consistent with the general principles of highly scalable distributed computing. However, due to the many implementations and interconnects for MPI, it will be extremely difficult for the users to provide prebuilt static binaries. If the code is built on the target resource, defaults from the compilers will increase the chance of success, since MPI applications are especially sensitive to compilers, compiler options, and the MPI environment. Interactive logins via GSI-SSH or some other solution to this problem must be developed.

Testcase

For our test case, the Weather Research and Forecasting (WRF) model was chosen. The reasons for choosing WRF include: 1) We wanted to use a real model with its associated complexities instead of a simple MPI test program; 2) The Engagement team has a number of users interested in large scale WRF runs and ensembles; 3) The WRF model has a large user base and is one of the more difficult MPI models from the scientific community to tune and operate; 4) WRF has a non-trivial build, including a dependency

on a Fortran 90 compiler. Intel or Portland Group compilers are most commonly used to build WRF. We also selected a group of sites at which to run the model.

Site: Purdue Caesar

The Purdue Caesar machine is an SGI Altix shared memory machine with 128 processor cores and 512 GB of memory. Being a shared memory machine, the OpenMP version of parallel applications like WRF is best suited for this machine. For the case of WRF, we used the WRF version compiled with OpenMP options, which was pre-installed on $OSG_APP/WRFV2. Even though we could start up WRF execution using this version and the data set from our UC Davis collaborator, we could not manage a full run because the pre-installed version did not support "nesting" (WRF can be run with multiple options that can be configured by changing the namelist files). However, simple OpenMP applications ran fine on this machine.

We used the following procedure to run applications on the Purdue Caesar machine. We used the OSG software stack on the submit machine to submit a Condor Globus (gt-2 universe) job on the head node of Caesar (julius.rcac.purdue.edu) using the "fork" jobmanager. This job is essentially a Perl script that runs on the head node of Caesar. The script is responsible for:

1. Transferring the required input files for the application – It determines the "gassurl" and uses globus-url-copy to transfer the input files from the submit machine to the current directory on Caesar
2. Transferring the PBS submit file from submit machine to Caesar using globus-url-copy.
3. Submitting the PBS job using qsub.
4. Waiting for the job to terminate (This is done by grep-ing for a particular string in the output, for eg. "SUCCESS COMPLETE WRF" for WRF)
5. Transferring the output files back to the submit machine using globus-url-copy.
6. Deleting the files in the current execution directory on Caesar. (It is important to clean up after execution)

Although not suitable, there is an MPI environment for this machine. We were able to run simple MPI jobs on this machine by submitting a Condor gt-2 universe job on the head node of Caesar using the "fork" jobmanager. This job is a Perl script that submits a PBS MPI job. The following needs to be done in the PBS submit file in order to run a MPI job on Caesar. First, the mpich2-intel module needs to be loaded using a command like "eval `/opt/modules-3.1.6/$MODULE_VERSION/bin/modulecmd tcsh load mpich2-intel`". Next, one needs to run "mpdboot" on the nodes in $PBS_NODEFILE to start the daemons using a command like "/opt/mpich2-1.0.5/64/shm-intel-9.1.049/bin/mpdboot -f $PBS_NODEFILE". Then, one can issue the "mpirun" command to actually start the MPI application. For example, "mpirun -np 16 /autohome/u102/amandal/tmp/testPBS/testMPI" runs 16 test MPI processes on 16 cores of the machine.

The main impediment for running WRF via the grid interfaces is not having the right version of the binary compiled on the machine. An important point here is that different users often require different versions of WRF. Also, remotely debugging an application run is very difficult. It is nearly impossible to ship a precompiled version of a parallel code because of differences in compilers and runtime environments. The solutions may be (a) either have a library of well-tested WRF installations and advertise their configurations and runtime environments or, (b) have compilation support from the remote site administrators or, (c) have a way to remotely compile the parallel application (very difficult).

Our experience with the Purdue Caesar site has been good. The Grid services were up most of the time. The site administrators provided great support. The machine problems were fixed quickly and administrators were responsive to the users. To improve this site, we need improved application and compilation support.

Site: Purdue Lear

Lear has a full set of Intel compilers available so we opted to build WRF on the compute element. The system administrator, Preston Smith, was very accommodating and he configured and tested the jobmanager for (jobtype=mpi) operation. Discovering details on the MPI setup was done over email. The available MPI version is MPICH 1.2.7 with TCP/IP interconnect. Running jobs with (jobtype=mpi) seemed to work well at first, but then we noticed that only one process was started. We believe this has to do with the environment problem mentioned earlier. The problem is currently under investigation.

One thing to note is that (jobtype=mpi) jobs usually require some extra RSL. In the case of Purdue, the extra RSL was (queue=standby)(xcount=2)(host_xcount=8). The final Condor submit file was:

```
universe        = grid
grid_type       = gt2
globusscheduler = lepton.rcac.purdue.edu/jobmanager-pbs
globusrsl =
(jobType=mpi)(queue=standby)(xcount=2)(host_xcount=16)(directory=/scrat
ch/osg/engage/wrf/testcase)

executable = /scratch/osg/engage/wrf/testcase/wrf.exe

stream_output = False
stream_error  = False

WhenToTransferOutput = ON_EXIT
TransferExecutable = False

output = out/Purdue-Lear.out
error = out/Purdue-Lear.err
log = out/Purdue-Lear.log

notification = NEVER
queue
```

The extra RSL is another example of what the sites would need to advertise when supporting MPI jobs. The necessary parameters will most likely not be the same across sites, which presents an opportunity for OSG.

Site: Buffalo

GRASE-CCR-U2 was chosen as a site which we would try to get jobs working without interacting with the system administrators. We wanted a test case which would mimic an everyday user trying to get their job working on a site by only reading OSG and site documentation.

The first thing we tried was to submit a shell script to the fork jobmanager, which created a PBS job script, and then submitted it to the local scheduler with qsub. The main problem at this site was the probing. Remember, we did not want any email communication for this test case. After some probing (that is, lots of fork jobs) we figured out that the MPI implementation installed was LAM based, so we had to add lamboot and lamhalt to the PBS script. Once this was taken care of, simple MPI jobs worked fine. We were not able to get WRF running. We think it has to do with our statically linked WRF binary, but further investigation will be needed to be certain.

GRASE-CCR-U2 also has a (jobtype=mpi) interface configured, including using lamboot and lamhalt to boot/halt the MPI environment. Similar to the case above, we were not able to provide a statically built WRF binary to match the version installed on the system.

Site: GLOW

GLOW was chosen because we learned that local users were successfully running MPI codes. Dan Bradley and Greg Thain, the local system administrators, helped us get going by providing necessary information about their system, scripts, and a new install of MPI on their system. We built a statically linked WRF binary matching the MPICH2 version on the GLOW system.

GLOW's local scheduler is Condor. The MPI environment is controlled by MPD, the MultiProcessing Daemon, which is yet another method of booting up and providing an MPI environment. A wrapper script was needed to boot the MPD environment using Condor environment variables.

Because we knew a wrapper script was needed, we once again used the trick of submitting a fork job to submit the wrapper to the local scheduler using condor_submit. We had a couple of failed runs due to confusion on what MPI version was installed on the system, but after tweaks and a couple rebuilds, we had a binary matching the system MPI. A successful, albeit small because of the TCP/IP interconnect, WRF run was completed. (jobtype=mpi) does currently not work at GLOW, but there is no reason the wrapper script above could not be integrated with the Globus jobmanager Perl module.

Site: NERSC

NERSC is a site which requires OSG users to hold NERSC user accounts, so with that, we could login to the NERSC OSG resources and work there locally. This was extremely convenient for compiling the WRF code locally there and in addition, provided access to the NERSC helpdesk consultants, who were an extremely competent and helpful group. In fact, we were constrained to logging in at NERSC also for running (not just compiling) the WRF code as a parallel job, since their machines which support MPI did not (at the time of the testing) have a Globus gatekeeper interface. Gatekeepers for some of these machines are being implemented this summer, so this situation is evolving. Although we cannot yet comment of the level of support for MPI jobs at this site via interfaces implemented in the OSG software stack, we have documented our activities at NERSC in this section of the report.

For the runs, the WRF codes were built on two NERSC platforms: seaborg (an older IBM Power3 and slower but cheaper to use) and jacquard (a faster Opteron cluster, more expensive to use, with much better throughput). The differences in the machines – their environments, their MPI implementations, etc – did not weigh heavily, thanks to the remarkable documentation furnished by NERSC. It was very clear how to use the MPI optimally for a specific architecture – as it was all explained in the documentation, machine by machine.

The WRF runs, covering 48 hours of simulated time, took around 2 hours to complete on 48 CPUs of jacquard and around 10 hours to complete on 48 CPUs of seaborg. We ran 16 jobs a day - ensembles of WRF runs - in order to furnish the data for a comparative study of the various physics models in the code. The study, "Probabilistic QPF using a multi-physics WRF ensemble", was presented at the 22nd Conference on Weather Analysis and Forecasting/18th Conference on Numerical Weather Prediction in Park City, UT, June 25-29. A paper is now in preparation. The runs continued for about 25 days, using a total of around 180,000 hours of the OSG allocation at NERSC. The collection of WRF forecasts has already allowed a systematic, in-depth investigation of precipitation predictability at small spatial scales and has shed light on how variations in initial conditions, soil physics scheme, boundary layer parameterization, and representation of moist processes in the atmosphere (vapor, liquid, water, ice, etc.) impact the prediction of precipitation amount and location.

As far as the NERSC site itself, it is so well attuned to the needs of a wide user community that its MPI implementations, job submission environments, data storage access, etc (essentially all the details a new user needs to get up and going at this site) are wonderfully, clearly, thoroughly documented, machine by machine. We never encountered a problem with unknown MPI environment details, which vary by platform, because the NERSC documentation makes the use of their tools and libraries exceedingly clear for each machine. When we did hit a glitch, from $SCRATCH quotas to problems with the HPSS allocation to requesting more time on the OSG allocation, without fail the NERSC consultants responded quickly and with useful information and solutions. We were really very impressed with their dedication to their users, and our meteorological study certainly benefited from that!

As far as improvements for OSG, this site should simply continue with its planned implementation of the Globus gatekeepers for the platforms that support MPI.

Recommended Actions

There are a set of actions which could transform OSG into a great provider of services for MPI jobs. The changes include software stack improvements, configuration, and advertisement of MPI capabilities and configurations. Initially, we recommend:

- Preserving the environment from the local scheduler when adding the OSG environment so the job can setup on multiple nodes
- Sites that want to serve MPI users should maintain (jobtype=mpi) interfaces, and testing of the interface should be performed by the GOC
- Sites should advertise MPI availability, location, implementation, transport, necessary RSL variables, and anything else required to run MPI jobs (CEMon?)
- Sites should ensure that a Globus gatekeeper node has an environment very similar to that of the usual front end and compute nodes of a compute resource, since having different environments makes compiling on the system difficult.
- A discussion should be started to determine if multiple MPI installs are needed per site. Traditionally, most HPC clusters have many installs. The question is how this should approached for Grid jobs.

Longer term, user feedback will be needed to further shape OSG's MPI offerings. The Engagement team is interested in following these developments and working with new user communities to drive these requirements and enhance the user experience for MPI jobs on OSG.

Summary

Improving the support of MPI applications on OSG is not a huge task, but it will likely take some time to converge on a common set of interfaces and conventions. We believe that the amount of work involved is well worth the time, as the end result will be an even more "Open, able to service more Sciences, Grid".

In the meantime, the Engagement VO will continue to work with users who want to run MPI jobs, and sites who want to provide early MPI access.