Univa Corporation

Grid Engine Documentation

Univa Grid Engine Cray XC30 / XC40 Integration

Author:
Univa Engineering

Version:
1.00

January 12, 2016

Contents

1 Cray XC30 / XC40 Integration for Univa Grid Engine

1.1 Definitions

1.1.1 login node

Cray XC30 / XC40 login node where the Cray tool **aprun** starts jobs in the Cray cluster. On this host Univa Grid Engine jobs are running and starting off compute jobs.

1.1.2 compute node

Cray XC30 / XC40 compute node. No Univa Grid Engine component is running there.

1.1.3 interactive node

Cray XC30 / XC40 interactive node on which no Univa Grid Engine job is allowed to run.

1.1.4 batch node

Cray XC30 / XC40 compute node where only jobs which are started on a login node in the Univa Grid Engine environment are running.

1.2 Required Files

- CrayBasil.py: Helper tool for the integration scripts used for communication with ALPS using BASIL and for Univa Grid Engine qstat parsing.
- UGE_prolog.sh: Prolog script for queues targeting a Cray XC30 / XC40. It creates an ALPS reservation on the Cray system for each job.
- UGE_epilog.sh: Epilog script for queues targeting a Cray XC30 / XC40. It cancels the ALPS reservation for a job and collects job run-time usage metrics.
- UGE_load_sensor.sh: Load sensor which updates "cray_nodes" when configuration is changed.
- UGE_JSV.tcl: The JSV script which modifies the Cray job parameters in a way that the cray_nodes are aligned to the reservation parameters (mpp* resources).
- UGE_Install_Cray.sh: Installation script, which creates the "cray.q" queue, the required parallel environment, and the resources (complexes).
- modify_execd_startup.sh: Scripts which is called by the installation script. Modifies the Univa Grid Engine execution daemon startup script. It might be possible that a Cray specific procedure (xtopview) needs to be followed to change the startup script manually.
- templates directory containing installation templates

The execution daemon and the shepherd must support creating the PAGG environment variable. This is not part of the official Univa Grid Engine binaries until and including version 8.1.7. If you want to run the Cray XC30 / XC40 integration in such versions, you need to exchange the execution daemon binary and the shepherd binary by the special builds delivered from Univa support.

1.3 System Overview

On each login node of the Cray XC30 / XC40 system Univa Grid Engine execution daemon can be installed. A Univa Grid Engine execution daemon must be installed on at least one login node.

Two types of Univa Grid Engine jobs can be executed on the login node:

- **Login node only jobs**: Examples of this kind of jobs are jobs which compile binaries for the Cray platform. Those jobs have to request the cray.q (-q cray.q or, equivalently, -l q=cray.q) but not the cray parallel environment and not any Cray resources (cray_nodes and/or mpp_* resources).
- **Cray XC30 / XC40 jobs**: Such jobs run applications on Cray compute nodes with **aprun**. For getting compute nodes, those jobs have to request: cray.q queue (-q cray.q or, equivalently, -l q=cray.q), the amount of compute nodes by using the "cray" parallel environment (-pe cray), optionally a specific layout of the Cray application can be requested by using "mppwidth", "mppdepth", or "mppnppn". If not any specific layout is requested "mppwidth" is set to the amount of requested "slots" and "mppnppn" is set to "1", i.e. that on each compute node one processing element (instance of the binary) is started up by **aprun**. Examples can be found in the "Job submission" section.

Only "exclusive jobs" are running in the Cray XC30 / XC40 environment, i.e. that each "slot" of the parallel Cray job corresponds to one host. Only one job can run on a Cray compute node at a time. The "cray_nodes" global consumable tracks the amount of free Cray compute nodes.

A Cray job is started by the Cray "aprun -B" command within the job script, which is executed on the login node. The binary to be executed on the Cray system must be accessible on the login node. The "-B" parameter means that **aprun** runs the application in the way the reservation is done. The reservation parameters can be seen in the qstat -j output (mpp* parameters). Those are set/validated or corrected by the JSV script. Aprun can also be started with a different architecture than requested in the reservation, but only a subset of nodes from the reservation can be used. If more nodes are going to be used the **aprun** command will abort immediately.

A load sensor is installed (UGE_load_sensor) which continuously checks if "cray_nodes" resources are aligned to the amount of configured online "BATCH" nodes in the Cray XC-30 environment. It runs on only one login node. It might be neccessary to re-configure it when login nodes are changing. If the login node which runs the load sensor is going to be removed from the Univa Grid Engine cluster the load sensor must be started at a valid login node. This is done by setting the "load_sensor /UGE_load_sensor.sh" in the execution daemon configuration (qconf -mconf).

1.3.1 Job submission

A Cray XC30 / XC40 compute job is submitted in following format:

```
qsub -q cray.q -pe cray <amount of hosts> <your script which starts aprun -B>
```

The qmaster JSV script (*UGE_JSV.tcl*) evaluates the parameters. If a *cray.q* is requested it adds the required mpp_* resources to the job request. Those resources are used in the BASIL reservation for determining how many parallel instances (width) in which distance (distances) are

started. The special parameter **mppnppn** determines the amount of started instances per host. Additionally **cray_forced** is added as request. This requests protects *cray.q* from being used for non-Cray jobs.

The logic might be subject to change according the needs in the cluster. But in any case it is required that **cray_nodes** (as resource request) multiplied by requested PE slots equals the amount of nodes which are requested in the ALPS reservation. In other words each slot on the Cray XC30 / XC40 machine represents one node in the Univa Grid Enginesystem. One nodes can be either requested by using the **cray_nodes** resource and/or by requesting parallel environment slots.

When the job can run in the Cray cluster according to the free amount of the **cray_nodes** resources then the Univa Grid Engine job script is transferred by the Univa Grid Engine environment to the Cray login node. There the UGE_prolog.sh script is started. It reads out the mpp_* resource requests and creates a Cray ALPS reservation through the BASIL using a job container which is created in the Univa Grid Engine shepherd process.

After the prolog script running successful the job script is started. Now every **aprun** uses the reservation implicitly, i.e. **aprun** can not use more/other hosts than granted. Nevertheless multiple **aprun** can run sequentially in the job script using the same set of nodes.

After the job ends, the epilog script runs. It cancels the reservation until it disappears. It gathers accounting from the Cray RUR interface and puts it into the Univa Grid Engine accounting. The following values are forwarded (Cray format / Univa Grid Engine format):

- utime -> ru_utime
- stime -> ru_stime
- max_rss -> ru_maxrss
- wchar -> ru_oublock
- rchar -> ru_inblock

They will appear in the Univa Grid Engine accounting file. The cpu value in Univa Grid Engine accounting will be utime + stime. Please be aware that Cray RUR currently does not support online accounting. Hence cpu time in qstat is not representative of the CPU usage incurred by the job.

Example 1

Submit a job script which starts up 3 times the worker.sh script on three different hosts.

```
> qsub -q cray.q -pe cray 3 /home/crayadm/job_script.sh
Your job 279 ("job_script.sh") has been submitted

> qstat -j 279
...
hard resource_list:         cray_forced=1,cray_nodes=1,mppwidth=3,mppdepth=32,mppnppn=0
```

Here you can see that the prolog is doing a BASIL reservation with a “width” of 3 (this means it starts 3 instances) and a depth of 32 (which means each instance has 32 cores, i.e. one host). “nppn” - the amount of instances per node is 0 and hence ignored.

The Cray system shows following information for the reservation:

```
> apstat -a
Total placed applications: 1
Apid ResId    User PEs Nodes   Age State   Command
3824  4262 crayadm   3     3 0h00m   run worker.sh
```

Here 3 Cray PEs (which are parallel instances of the worker.sh script) on 3 nodes are reserved and started.

Example 2

Submit a job script which starts up 16 times the worker.sh script, with a distance of 4 cores on 32 core hosts (i.e. the script is started 8 times on each of the two hosts).

```
> qsub -q cray.q -pe cray 2 -l mppwidth=16,mppdepth=4 /home/crayadm/job_script.sh
Your job 288 ("job_script.sh") has been submitted

> qstat -r
288 0.55500 job_script crayadm      r     01/17/2014 02:47:09 cray.q@nid00034
    Full jobname:     job_script.sh
    Master Queue:     cray.q@nid00034
    Requested PE:     cray 2
    Granted PE:       cray 2
    Hard Resources:   mppwidth=16 (0.000000)
                      mppdepth=4 (0.000000)
                      cray_forced=1 (0.000000)
                      cray_nodes=1 (200000.000000)
    Soft Resources:
    Hard requested queues: cray.q
    Binding:          NONE

> apstat -r
ResId ApId From      Arch PEs N d Memory State
   4296 3850 batch:288   XT  16 0 4   1024 conf,claim
A  4296 3851 batch:288   XT  16 - -   1024 conf,claim

> apstat
Compute node summary
    arch config     up   resv    use  avail   down
    XT     24     23      2      2     21      1
No pending applications are present
Total placed applications: 1
Apid ResId    User PEs Nodes   Age State   Command
3851  4296 crayadm  16     2 0h00m   run worker.sh
```

Example 3

Submit a job with a width of 4 (mppnwidth=4, amount of instances) and having started 2 instances per host (mppnppn=2). Here 3 nodes are requested through the cray parallel environment, which is wrong and therefore corrected by the JSV script.

```
> qsub -q cray.q -pe cray 3 -l mppwidth=4,mppnppn=2  /home/crayadm/job_script.sh
Your job 295 ("job_script.sh") has been submitted

> apstat
Compute node summary
 arch config     up   resv    use  avail   down
   XT     24     23      2      2     21      1

> qstat -j 295
...
account:                    sge
hard resource_list:         mppwidth=4,mppnppn=2,cray_forced=1,mppdepth=1,cray_nodes=1
mail_list:                  crayadm@nid00034
notify:                     FALSE
job_name:                   job_script.sh
jobshare:                   0
hard_queue_list:            cray.q
env_list:
script_file:                /home/crayadm/job_script.sh
parallel environment:  cray range: 2
context:                    reservation_1=4303
...
```

Example 4

Same as example 3 but without requesting the cray parallel environment. Here the JSV script adapts the "cray_nodes" request so that it reflects the amount of actually reserved Cray nodes.

```
> qsub -q cray.q -l mppwidth=4,mppnppn=2  /home/crayadm/job_script.sh
Your job 297 ("job_script.sh") has been submitted

> qstat -j 297
...
hard resource_list:         mppwidth=4,mppnppn=2,cray_forced=1,mppdepth=1,cray_nodes=2
...

> apstat -r
ResId ApId From      Arch PEs N d Memory State
      4305 3868 batch:297   XT   4 2 1   1024 conf,claim
   A  4305 3869 batch:297   XT   4 - -   1024 conf,claim
```

Example 5

It shown an interactive jobs which starts up the parallel Cray job multiple times on command line. The last request does not match with the BASIL reservation, hence it is rejected.

```
> qrsh -pe cray 2 -q cray.q

# now on login node
```

```
> aprun -B /home/crayadm/installation/examples/jobsbin/lx-amd64/work 10
Forking -1 times.
Forking -1 times.
Application 3872 resources: utime ~200s, stime ~0s, Rss ~3572, inblocks ~0, outblocks ~0

# using the reservation requests: -B
> aprun -B /home/crayadm/installation/examples/jobsbin/lx-amd64/work -w 5
Forking -1 times.
Forking -1 times.
Application 3876 resources: utime ~10s, stime ~0s, Rss ~3572, inblocks ~0, outblocks ~0

> aprun -n 2 -d 32 /home/crayadm/installation/examples/jobsbin/lx-amd64/work -w 5
Forking -1 times.
Forking -1 times.
Application 3877 resources: utime ~10s, stime ~0s, Rss ~3572, inblocks ~0, outblocks ~0

> aprun -n 3 -d 32 /home/crayadm/installation/examples/jobsbin/lx-amd64/work -w 5
apsched: claim exceeds reservations node-count
```

1.3.2 Supported Job Types

Batch jobs: qsub -q cray.q -pe cray “amount of hosts” “job_script”

Interactive jobs: qrsh -q cray.q -pe cray “amount of hosts”

Array jobs: qsub -t N-M -q cray.q -pe cray “amount of hosts” “job_script”

With interactive jobs a shell on a Cray login node is opened where you can run **aprun** interactively.

1.3.3 Resources

Following resources are installed in Univa Grid Engine complex configuration.

Complex	**Semantic**
cray_nodes	Global consumable resource for compute node limitation
cray_alps_version	Displays ALPS version detected at installation time
cray_basil_version	Displays BASIL version detected at installation time
cray_cores_per_host	Displays the amount of cores on a Cray compute node
cray_forced	This complex protects cray.q so that no other job runs in
cray_numa_nodes	Amount of NUMA nodes on a Cray compute node
mppwidth	Cray reservation request: Amount of Cray PEs to start up
mppdepth	Cray reservation request: Distance between two Cray PEs
mppnppn	Cray reservation request: Amount of Cray PEs on a node

Table 1: Cray XC30 / XC40 related complexes

1.4 Installation

Requirements:

- Univa Grid Engine execution daemons installed on the Cray login nodes
- List of the Cray login nodes being present

Procedure:

- Switch to Univa Grid Engine admin user or root on a Univa Grid Engine admin host
- Source settings.sh or settings.csh file to add the required Univa Grid Engine environment variables to your environment.
- apbasil must also be accessible in the path
- cd $SGE_ROOT/util/resources/cray_xc30_integration
- Test CrayBasil.py with ./CrayBasil.py –nodes (should return the amount of nodes)
- Adapt cray_settings.sh ! -> Set LD_LIBRARY_PATH to where the Cray libjob.so is installed (Hint: do a “module load job” on command line and check the path) -> Set normal PATH that **aprun** can be found (Hint: check the PATH on command line of a login node where **aprun** works)
- Execute ./UGE_Install_Cray.sh and follow instructions
- The execution daemon startup script needs to be adapted so that the libjob.so can be found by the shepherd. This is crucial for creating the PAGG environment variable needed for confirming the reservation in Univa Grid Engine prolog script. (Hint: change #!/bin/sh to #!/bin/bash -l in the sgeexecd init.d startup script and insert a “module load job” before starting the execution daemon).

Post Installation Check:

- Submit a “qrsh -q cray.q -pe cray 2” and try to run “aprun -B sleep 10” manually.

1.5 Special Topics

1.5.1 Job Suspension/Resume:

Job suspension and resumption should not be used on Cray jobs. Since jobs are running exclusively on the nodes, there is also no requirement for doing this.

1.6 Job Status:

During job run-time, the Cray XC30 / XC40 system does not collect any resource usage information, which can be forwarded to Univa Grid Engine. The values seen in the mem/io/cpu fields of qstat are reflecting the job script which launches the Cray job via **aprun** only.

1.7 Problem Solving

Due to certain circumstances (Cray ALPS reservation can not be done, timeouts etc.) jobs can enter the error state “E”. The error can just be temporarily due to Cray cluster / compute node changes. The Univa Grid Engine prolog and epilog scripts are printing detailed information for tracing into the jobs output and error file.

If the queue instance went into an error state due to a more severe circumstance it should be checked where the error comes from:

- qstat -explain E shows from which job the error originates
- the output files of the job need to be analyzed