Slurm Fault Tolerant Workload Management

19 October 2013
David Bigagli, Morris Jette
[david,jette]@schedmd.com

SchedMD LLC
http://www.schedmd.com

Motivation

- Failures in large computers are inevitable.
- Bigger the size of the parallel job higher is the probability of failure at runtime.
- Implement failure recovery services in Slurm which can be used by running applications to respond to observed or anticipated failures.

Current approach

- Workload manager based:
  - Using re-runnable jobs
  - Using job dependencies
  - Allocating extra resources
- Application based:
  - System-level checkpointing
  - Application-level checkpointing
- If failures are common, the impact on application performance is significant

Helping applications be resilient
(Dr. William Kramer, NCSA, SUG 2011)

- Applications are able to reallocate work
- The resource manager provides assistance to substitute resource with just in time delivery
- A protocol between resource manager and application to negotiate the best solution:
  - System: “Your node x just broke”
  - App: “Can I have another node to replace it”
  - System: “Yes, but not for 50 minutes”
  - App: “Ok then just drop it and extend my runtime”

Helping applications be resilient
(Dr. William Kramer, NCSA, SUG 2011)

- Another example:
  - App: “My node Y is not responding”
  - System: “I can give you another one in 5 minutes”
  - App: “Can you make it 2 nodes so I can make up the lost time”
  - System: “Yes, but not for 7 minutes”
  - App: “Also adjust my time limit by 20 minutes”
  - System: “I have something else waiting, but can give you 10 minutes”
  - App: “Ok”

Slurm failure management infrastructure

- Failed hosts, currently out of service
- Failing hosts, malfunctioning and/or expected to fail
- Hot Spare
  - A cluster-wide pool of resources to be made available to jobs with failed/failing nodes
  - The hot spare pool is partition based, the administrator specifies how many spares in a given partition
  - Any node in the partition can be part of the spare pool
  - Access control list to the hot spare indicating which user/group may or may not use it

Slurm failure management infrastructure

- Failing hosts can be drained and then dropped from the allocation, giving application flexibility to manage its own resources.
- Drained nodes can be put back on-line by the administrator and they will go automatically back to the spare pool.
- Failed nodes can be put back on-line and they will go automatically back to the spare pool.

Slurm failure management infrastructure

- Application usually detects the failure by itself, losing one or more of its component
  - Able to notify Slurm of failures and drain nodes
- Application can also query Slurm about state of nodes in its allocation
- Application asks Slurm to replace its failed/failing nodes, then it can
  - Wait for nodes become available, eventually increasing its runtime till then
  - Increase its runtime upon node replacement
  - Drop the nodes and continue, eventually increasing its runtime

Slurm architecture

- Slurmctld plugin keeps track of the spare pool and the job allocation status
- libsmd.so, libsmd.a and smd.h are the client interface and library to the non stop services based on a jobID
- snonstop command build on top of the library provides command line interface to the recovery services.
- nonstop.sh shell script which automates the node replacement based on user supplied environment variables.

SnonStop usage

- The application runs unchanged and at every step it checks the health of its nodes.
- The application may link with the libsmd.so library and use the nonstop API to retrieve the node status and take action to replace them.
- The application may link with the libsmd.so library and subscribes for events which will be delivered asynchronously.
- The job has to be submitted to Slurm using the --no-kill option to prevent it being killed upon component failure.

SnonStop common use case

```
#!/bin/sh

# Set the environment variable to handle
# runtime node failure.
export
SMD_NONSTOP_FAILED=REPLACE:TIME_LIMIT_DELAY=10:EXIT_JOB
  i=0
  while [ $i -le 100 ]
  do
  # Run the $i step of my application
    srun myapp
  # Detect failure and execute actions
    nonstop.sh
    if [ $? -ne 0 ]; then
       exit 1
    fi
    let i=i+1
  done
```

SnonStop configuration

- Environment variables to determine the nonstop action to recover nodes
- SMD_NONSTOP_FAILED or SMD_NONSTOP_FAILING =
  - REPLACE:DROP:EXIT_JOB
  - TIME_LIMIT_DELAY
  - TIME_LIMIT_EXTEND
  - TIME_LIMIT_DROP

nonstop.conf

```
#
ControlAddr=prometeo
#
Debug=0
Port=34000
UserDrainAllow=david
#UserDrainDeny=david
#
# Extend time upon node replacement
TimeLimitExtend=15
# Extend time upon node drop
TimeLimitDrop=22
# Extend time while attempting to replace node
TimeLimitDelay=12
#
HotSpareCount=bootes:2
#
MaxSpareNodeCount=2
```

Example of use

- Termination of a component of the parallel job causing the step to abort
- If there are enough nodes the replacement is automatic

```
srun: error: achab5: tasks 16-23: Killed
is_failed: job 130 searching for FAILED hosts
is_failed: job 130 has 1 FAILED node(s)
is_failed: job 130 FAILED node achab6 cpu_count 8
_handle_fault: job 130 handle failed_hosts
_try_replace: job 130 trying to replace 1 node(s)
_try_replace: job 130 node achab6 replaced by achab7
_generate_node_file: job 130 all nodes replaced
source the /tmp/smd_job_130_nodes.sh hostfile to get the new job environment
_try_replace: job 130 all nodes replaced all right
```

Discussion

- Question and answers.
- Thank you!