N1GE6 Checkpointing and Berkeley Lab Checkpoint/Restart

Liang PENG
Lip Kian NG

N1GE6 Checkpointing and Berkeley Lab Checkpoint/Restart

Liang PENG
Lip Kian NG

APSTC-TB-2004-005

Abstract:

N1GE6, formerly known as Sun Grid Engine, is widely used in HPTC environment for efficient utilization of compute resources. As applications in such environment are generally compute intensive, fault tolerance is required to minimize the impact of hardware failure. N1GE6 has several fault tolerance features and in this report, the focus will be on the checkpointing support and the integration of Berkeley Lab Checkpoint/Restart will be used as an example.

Keywords: checkpoint, Grid Engine, blcr

Email Address:
pengliang@apstc.sun.com.sg
lkng@apstc.sun.com.sg

Revision History

Version	Date	Comments
1.1	Jul 14, 2004	
1.2	Dec 28, 2004	Feedback from Reuti (reuti__at__staff.uni-marburg.de) • Transparent interface is user-level (Table 1). • Update to state diagram (Illustration 2).

Asia Pacific Science and Technology Center
50 Nanyang Avenue, N3-01-c10
Singapore 639798

N1GE6 Checkpointing and Berkeley Lab Checkpoint/Restart

Liang PENG
Lip Kian NG
Asia Pacific Science and Technology Center
Sun Microsystems Pte Ltd, Singapore

Introduction

Checkpointing is the process of writing out the state information of a running application to physical storage periodically. With this feature, an application will be able to restart from the last checkpointed state instead of from the beginning which would have been computationally expensive in HPTC environment.

In general, checkpointing tools can be classified into 2 different classes:

- Kernel-level
  - Such tools are built into the kernel of the operating system. During a checkpoint, the entire process space (which tends to be huge) is written to physical storage.
  - The user does not need to recompile/re-link their applications.
  - Checkpointing and restarting of application is usually done through OS commands.
  - Checkpointed application is usually unable to be restarted on a different host.
- User-level
  - These "tools" are built into the application which will periodically write their status information into physical storage.
  - Checkpointing of such applications is usually done by sending a specific signal to the application.
  - Restarting of such applications is usually done by calling the application with additional parameters pointing to the location of restart files.

N1 Grid Engine (N1GE6) Checkpointing and Migration Support[1]

N1GE6 has built-in support for the integration of 3rd party checkpointing tools. Certain checkpointing tools (mostly user-level) allow the restart of applications on different hosts. These tools, coupled with the migration support on the N1GE6 and with proper configuration of the queue threshold levels, allows the administrator to finely load balance the N1GE6 cluster.

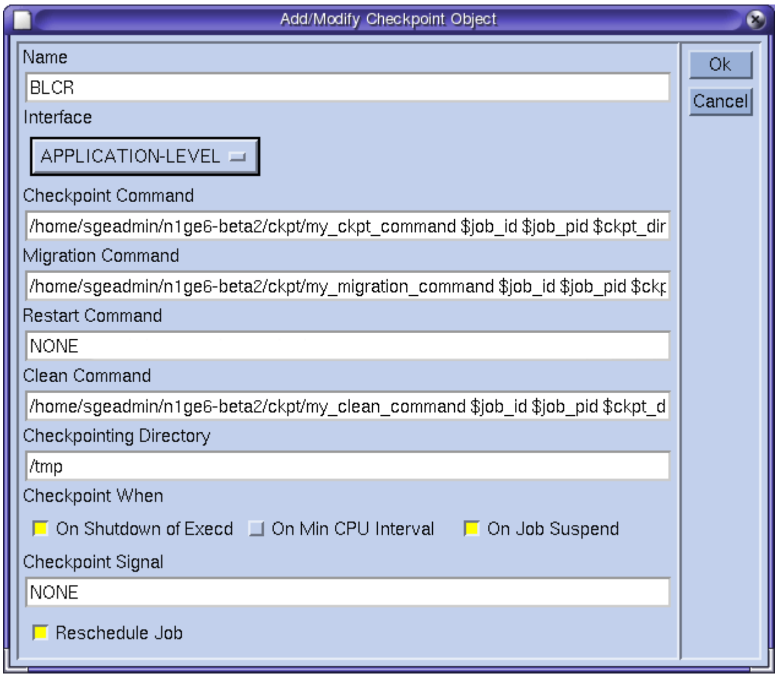
The following sections will show the checkpoint and migrate support on the N1GE6.

Illustration 1 shows the N1GE6 checkpoint configuration menu.

Interface	6 different interfaces are available and these will determine which of the following commands are used. (See Table 1)
Checkpoint Command	Path to script which will be executed by N1GE6 to initiate a checkpoint.
Migration Command	Path to script which will be executed by N1GE6 to initiate a migration.
Restart Command	Path to script which will be executed by N1GE6 to restart the job.
Clean Command	Path to script which will be executed to initiate the cleaning up of a checkpoint job. (Eg. Deletion of restart files)

[1] N1GE6 does not provide any checkpointing tools but has built-in support for the integration of 3rd party tools.

Checkpoint When	3 options: • On Shutdown of Execd – If possible, checkpoint, abort and migrate the job if the corresponding execution host shuts down the execd daemon. • On Min CPU Interval – Checkpoint the job periodically after it has executed for a pre-defined CPU time interval (this interval is defined under queue_conf). • On Job Suspend – Checkpoint, abort and migrate the job if the job is suspended either through user intervention or threshold exceed.
Checkpoint Signal	Unix signal to be sent to the job to initiate a checkpoint.
Reschedule Job	When selected, the job will be rescheduled (ie. restarted) instead of checkpointed when the execution host goes into unknown status.



Name
BLCR

Interface
APPLICATION-LEVEL

Checkpoint Command
/home/sgeadmin/n1ge6-beta2/ckpt/my_ckpt_command $job_id $job_pid $ckpt_dir

Migration Command
/home/sgeadmin/n1ge6-beta2/ckpt/my_migration_command $job_id $job_pid $ckp

Restart Command
NONE

Clean Command
/home/sgeadmin/n1ge6-beta2/ckpt/my_clean_command $job_id $job_pid $ckpt_d

Checkpointing Directory
/tmp

Checkpoint When
☑ On Shutdown of Execd □ On Min CPU Interval ☑ On Job Suspend

Checkpoint Signal
NONE

☑ Reschedule Job

Illustration 1 N1GE6 Checkpoint Menu

Class	***Interface Name***	***Remarks***
Kernel	HIBERNATOR	All commands used.
	CRAY-CKPT	
	CPR	
User	APPLICATION-LEVEL	Restart command not used.
	USER DEFINED	All commands not used.
	TRANSPARENT	

Table 1. Interface properties

Job State Transition

Illustration 2 shows the state transition diagram for the different types of checkpointing interfaces and when each of the 4 commands are executed.

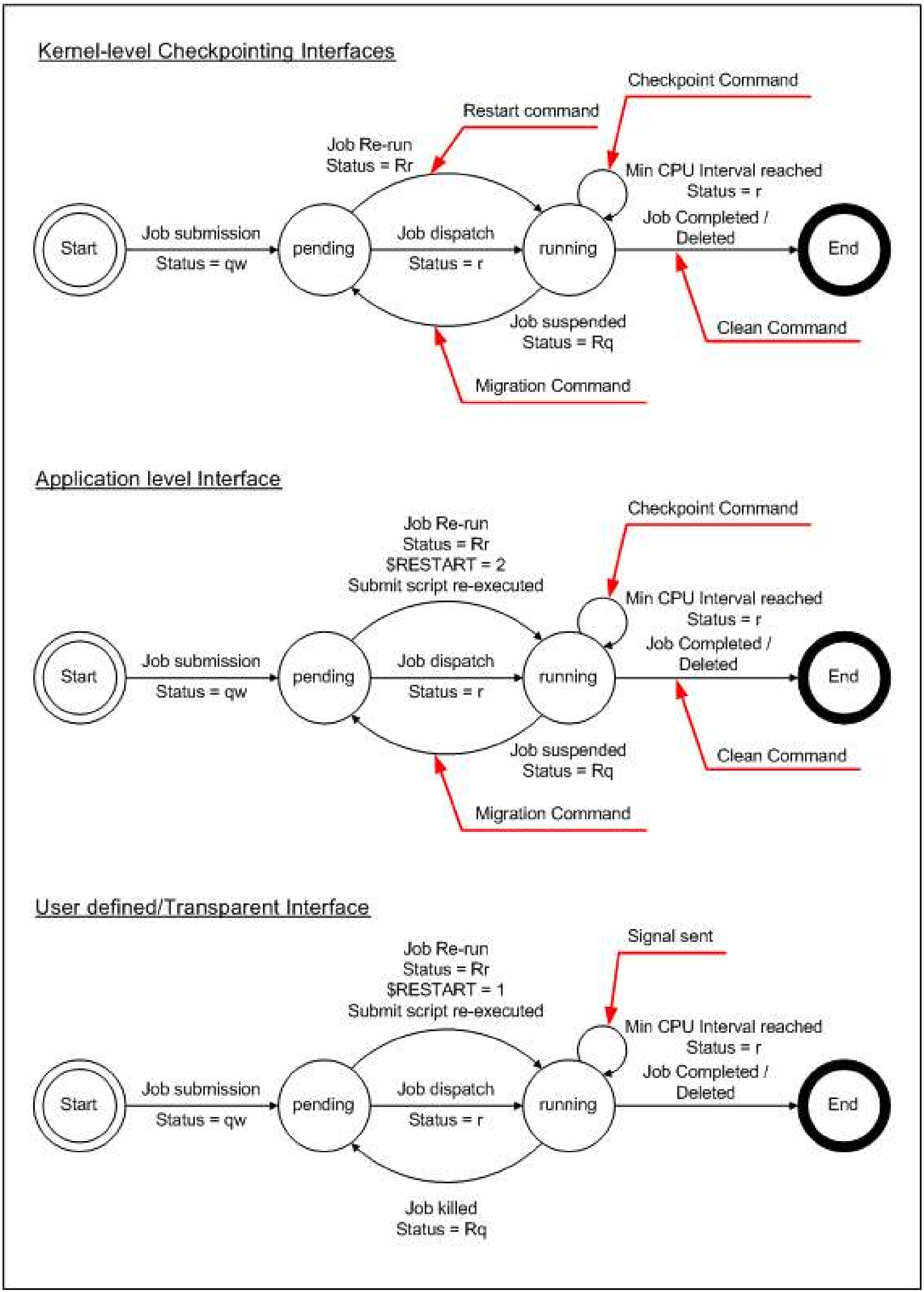


Illustration 2 State diagram of the different interfaces

Berkeley Lab Checkpoint/Restart

“Berkeley Lab Checkpoint/Restart (BLCR) is a kernel module that allows you to save a process to a file and restore the process from the file. This file is called a *context file*. A context file is similar to a core file, but a context file holds enough information to continue running the process. A context file can be created at any point in a process's execution. The process may be resumed from that point at a later time, or even on a different workstation.”

-- Future Technologies group

Basic Usage

A basic understanding on how to use BLCR is required to follow through the remaining sections.

Starting an application	cr_run <app name>
Checkpoint and continue	cr_checkpoint -f <context filename> <process id>
Checkpoint and terminate	cr_checkpoint -f <context filename> --kill <process id>
Restart	cr_restart <context file>

BLCR Known Limitations

1. BLCR doesn't support checkpointing of a process group yet.
2. To restart from a context file, the PID of the original process must NOT be in use.
3. To restart from a context file, the original executables and shared libraries used must exists and contents remain the same.

As a result of limitation 2 and the fact that process IDs are not unique between nodes in a cluster, the integration discussed below will not migrate jobs between 2 different nodes. (Note: a little trick can be used here by checking the status of the cr_restart command within the shell script and resubmitting the job. Find out more below.)

Integration of BLCR with N1GE6

Approach

Even though the BLCR is a kernel level checkpointing tool, its limitation on not being able to checkpoint a process group means that using kernel interfaces for integration may not be appropriate. The main reason is that during job restarts, the kernel interfaces restore the application process without reprocessing the submission script. Thus the ability to only checkpoint a single process meant that only one process within the script can be checkpointed and during job restart, only this process is restored and the control and data flow embedded within the shell script are lost.

As such, a simpler but workable approach is to integrate BLCR using the Application-level Interface. The difference between the Application-level Interface and any of the kernel-level interfaces is during job restarts, the submission scripts are re-executed and to differentiate between an initial job run and a restarted job, N1GE6 sets the internal variable $RESTARTED accordingly.

Now, since the submission scripts are re-executed on every restart and there is a method of determining if the job is restarted or not, additional logic can be added into the submission script to enable the checkpointed application to restart gracefully. (See Text 1)

Limitations

- There is only 1 binary for each submission script.
- LAM-MPI integration has not been done.

```
#!/bin/csh
if ( ${RESTARTED} ) then
  // code to restart the application
  cr_restart ...
  ...
else
  // code to start application
  cr_run ...
  ...
endif
```

Text 1 Sample submission script

Crafting the checkpoint script

The provided CPR checkpoint script is used as a starting point for the BLCR checkpoint script. The following discussion focuses on the core logic of the modified script. (See Appendix A for full source.)

```
82 ...
83 # get the pid of the running binary
84 cpid=`pstree -p $job_pid | awk -F "(" '{ print $NF }' \
        | awk -F ")" '{ print $1 }'`
85 /usr/local/bin/cr_checkpoint -f $ckptfile --run $cpid
86 ...
```

Text 2 checkpoint script

Recall from the discussion above that only one process can be checkpointed. However, since only shell scripts are permitted to be submitted to N1GE6, N1GE6 will have knowledge of the submission script's process id only (through the variable $job_pid). But since the submission script is really just a wrapper for the binary, the binary is essentially a child process of the submission script's process. So the purpose of $cpid (Text 2) is to retrieve the process id of the binary to be checkpointed.

Crafting the migration script

The provided CPR checkpoint script is used as a starting point for the BLCR checkpoint script. The following discussion focuses on the core logic of the modified script. (See Appendix A for full source.)

```
50 qalter -q $QUEUE $JOB_ID
...
83 # get the pid of the running binary
84 cpid=`pstree -p $job_pid | awk -F "(" '{ print $NF }' \
        | awk -F ")" '{ print $1 }'`
85 /usr/local/bin/cr_checkpoint -f $ckptfile --kill $cpid
86 ...
```

Text 3 migrate script

The only difference between the checkpoint and migrate script is in line 50 and 85 (Text 3). During job migration, the job is placed back in the pending state and can be re-scheduled by the N1GE6 scheduler. However, since uniqueness of process id is not possible between nodes, it is safer that migrating jobs do not migrate between nodes. Hence, line 50 ensures that when the migrated job is re-scheduled, it will only be scheduled on the same node (Line 50). At line 85, the migrating job will be killed after it has been checkpointed (since it is pointless for a migrating job to continue executing at this instance anymore.)

Crafting the clean script

The purpose of this script is to clean up the process and BLCR context files, hence there is nothing really interesting to discuss.

```
...
# workaround for qdel failing to kill restarted jobs
# make sure job is really dead
cpid=`pstree -p $job_pid | awk -F "(" '{ print $NF }' \
     | awk -F ")" '{ print $1 }'`
kill -9 $cpid >> $F 2>&1
kill -9 $job_pid >> $F 2>&1
...
```

Setting up the checkpointing environment in N1GE6

Step 1	Create the checkpoint environment
	`> qconf -ackpt BLCR`
	`ckpt_name       BLCR`
	`interface       APPLICATION-LEVEL`
	`ckpt_command    /n1ge6-beta2/ckpt/my_ckpt_command $job_id  \`
	`                $job_pid $ckpt_dir`
	`migr_command    /n1ge6-beta2/ckpt/my_migration_command  \`
	`                $job_id $job_pid $ckpt_dir`
	`restart_command none`
	`clean_command   /n1ge6-beta2/ckpt/my_clean_command $job_id  \`
	`                $job_pid $ckpt_dir`
	`ckpt_dir        /tmp`
	`signal          NONE`
	`when            xsmr`

Step 2	Attach the BLCR checkpoint environment to the queue.
	`> qconf -mq all.q`
	`...`
	`...`
	`qtype           BATCH INTERACTIVE`
	`ckpt_list       BLCR`
	`pe_list         make`
	`...`
	`...`

Conclusion

This report has detailed the checkpointing support of N1GE6 and the integration steps of BLCR into N1GE6. With the flexibility of N1GE6, the reader should be able to integrate most of the checkpointing tools available with some modifications to the checkpointing/migration/restart scripts.

Appendix A

```
#!/bin/sh

set +u
ckpt_dir=$3
if [ ! -f $ckpt_dir/ckpt.log ]; then
   touch $ckpt_dir/ckpt.log
   chmod 666 $ckpt_dir/ckpt.log
fi
sge_root=${SGE_ROOT}
sge_cell=${SGE_CELL}
# workaround to force job to restart on same queue (svd)
. $sge_root/${sge_cell:-default}/common/settings.sh

tmpdir=$ckpt_dir/ckpt.$1   # create temp dir for holding checkpoint info
mkdir -p $tmpdir
cd $tmpdir

# create log file
F=~/$REQNAME.co$1
touch $F
echo ----------------------------------  >> $F 2>&1
echo `basename $0` called at `date`      >> $F 2>&1
echo called by: `id`                     >> $F 2>&1
echo with args: $*                       >> $F 2>&1
echo on queue : $QUEUE                   >> $F 2>&1
# checkpoint the job to one of two different files (i.e. ping-pong)
# just in case we go down while checkpointing
currcpr=`cat currcpr`
if [ "$currcpr" = "2" ]; then
    currcpr=1
    prevcpr=2
else
    currcpr=2
    prevcpr=1
fi
ckptfile=context_$1.$currcpr
pid=$2
# get the child process to checkpoint
echo `pstree -p $pid` >> $F 2>&1
cpid=`pstree -p $pid | awk -F "(" '{ print $NF }' | awk -F ")" '{ print $1
}'`
echo Checkpoint command: cr_checkpoint -f $ckptfile --run $cpid >> $F 2>&1
/usr/local/bin/cr_checkpoint -f $ckptfile --run $cpid
cc=$?
if [ $cc -eq 0 ]; then
   echo $currcpr > currcpr
   if [ -f context_$1.$prevcpr ]; then
      echo Deleting old checkpoint file >> $F 2>&1
      # cpr -D cpr_$1.$prevcpr >> $F 2>&1
      rm -f context_$1.$prevcpr
   fi
fi

echo `date +"%D %T"` Job $1 "(pid=$cpid) checkpointed, status=$cc" >>
$ckpt_dir/ckpt.log
```

Text 4 blcr_checkpoint.sh

```
#!/bin/sh

set +u

ckpt_dir=$3

if [ ! -f $ckpt_dir/ckpt.log ]; then
   touch $ckpt_dir/ckpt.log
   chmod 666 $ckpt_dir/ckpt.log
fi
sge_root=${SGE_ROOT}
sge_cell=${SGE_CELL}
# workaround to force job to restart on same queue (svd)
. $sge_root/${sge_cell:-default}/common/settings.sh
qalter -q $QUEUE $JOB_ID

# create temp directory for holding checkpoint info
tmpdir=$ckpt_dir/ckpt.$1
mkdir -p $tmpdir
cd $tmpdir

# create log file
F=~/$REQNAME.co$1
touch $F
echo ----------------------------------  >> $F 2>&1
echo `basename $0` called at `date`      >> $F 2>&1
echo called by: `id`                     >> $F 2>&1
echo with args: $*                       >> $F 2>&1

# checkpoint the job to one of two different files (i.e. ping-pong)
# just in case we go down while checkpointing
currcpr=`cat currcpr`
if [ "$currcpr" = "2" ]; then
    currcpr=1
    prevcpr=2
else
    currcpr=2
    prevcpr=1
fi
ckptfile=context_$1.$currcpr

echo Migration command: cr_checkpoint -f $ckptfile --kill $cpid >> $F 2>&1
/usr/local/bin/cr_checkpoint -f $ckptfile --kill $cpid

cc=$?
if [ $cc -eq 0 ]; then
   echo $currcpr > currcpr
   if [ -f context_$1.$prevcpr ]; then
      echo Deleting old checkpoint file >> $F 2>&1
      #cpr -D cpr_$1.$prevcpr >> $F 2>&1
      rm -f context_$1.$prevcpr
   fi
fi

echo `date +"%D %T"` Job $1 "(pid=$cpid) checkpointed and killed,
status=$cc" >> $ckpt_dir/ckpt.log
```

Text 5 blcr_migrate.sh

```
#!/bin/sh
set +u

ckpt_dir=$3

if [ ! -f $ckpt_dir/ckpt.log ]; then
   touch $ckpt_dir/ckpt.log
   chmod 666 $ckpt_dir/ckpt.log
fi

# create temp directory for holding checkpoint info

tmpdir=$ckpt_dir/ckpt.$1
mkdir -p $tmpdir
cd $tmpdir

# create log file
#F=$tmpdir/checkpoint.log
F=~/$REQNAME.co$1
touch $F

echo ----------------------------------  >> $F 2>&1
echo `basename $0` called at `date`      >> $F 2>&1
echo called by: `id`                     >> $F 2>&1
echo with args: $*                       >> $F 2>&1

# workaround for qdel failing to kill restarted jobs
# make sure job is really dead

cpid=`pstree -p $2 | awk -F "(" '{ print $NF }' | awk -F ")" '{ print $1 }'`

kill -9 $cpid >> $F 2>&1
kill -9 $2    >> $F 2>&1

echo `date +"%D %T"` Job $1 "(pid=$cpid) cleaned up" >> $ckpt_dir/ckpt.log
```

Text 6 blcr_clean.sh

```
#!/bin/csh

set tmpdir=${SGE_CKPT_DIR}/ckpt.${JOB_ID}
set currcpr=`cat ${tmpdir}/currcpr`
set ckptfile=${tmpdir}/context_${JOB_ID}.$currcpr

if ( ${RESTARTED} && -e $tmpdir ) then
  echo "Restarting from $ckptfile" >> /tmp/restart.log
  /usr/local/bin/cr_restart $ckptfile
else
  /usr/local/bin/cr_run $*
endif
```

Text 7 Submission script

References

- N1GE6 User Manual
- N1GE6 checkpoint sample scripts
- Grid Engine Website (http://gridengine.sunsource.net/)
- Future Technologies Group (http://ftg.lbl.gov/checkpoint)