An Early Experience on Job Checkpoint/Restart - Working with SGI Irix OS and the Portable Batch System (PBS)

Yan-Tyng Sherry Chang
Scientific Consulting Group
NASA Advanced Supercomputing Division
NASA Ames Research Center, M/S 258-6
Moffett Field, CA 94035-1000

Email : schang@nas.nasa.gov
Tel : (650) 604-1272
Fax : (650) 604-4377

Abstract

In this paper, we describe the use of SGI's IRIX utility 'cpr' and the Portable Batch System (PBS) for job checkpoint/restart. The feasibility and reliability to checkpoint and restart using the cpr utility and/or PBS were tested. Specifically, four different procedures including (1) using cpr for interactive sessions, (2) using cpr in PBS scripts for batch jobs, (3) using PBS's qhold/qrls commands for batch control, and (4) using PBS's job submission (qsub) -c option for auto-checkpointing periodically were applied to two test cases. Results of these tests are presented.

Introduction

The ability to checkpoint and restart jobs is desirable for users who perform resource-intensive computation that takes a long time to run. Checkpointing prevents loss of valuable data that cannot be easily recovered when a system fails. Checkpoint/restart is also useful for system administrators in two scenarios. The first one is to improve job scheduling and load balancing of computer systems, and the second is to perform maintenance work such as replacing a piece of hardware without loss of users' jobs.

In general, the checkpoint/restart of a job can be pursued in two different ways. For a resource-intensive job, it is wise for code developers to provide built-in capability in their programs for checkpoint and restart. Examples of this approach are seen in various applications such as many of the Computational Fluid Dynamics codes and the well-known quantum chemistry package Gaussian that run on the NAS (NASA Advanced Supercomputing) computer systems. On the other hand, the operating system of a computer may also provide the capability to checkpoint and restart. At NAS, the Cray C90 computer using the Unicos operating system has been quite reliable for such a task when it needs to be shutdown. For SGI machines running with the Irix operating system, the cpr utility for checkpoint/restart are implemented in 6.n releases.

SGI's Origin 2000 series of machines have been used at NAS for several years. However, the feasibility and reliability of cpr have not been tested. Recently, in an effort to help with a particular user's case, where job checkpoint/restart can only be performed through the operating system, a series of testing on cpr and PBS was performed. In this paper, we present our recent experiences with checkpoint/restart using SGI's Irix cpr utility and the Portable Batch System (PBS). As will be shown in later sections, although checkpoint/restart succeeded in some cases, two caveats and five types of failures were found. Thus, our work shows that checkpoint/restart is still not reliable and more collaborative efforts are needed to ensure the feasibility and reliability of checkpoint/restart in real production systems.

The Irix Checkpoint/Restart Utility

As stated in the IRIX Checkpoint and Restart Operation Guide (007-3236-004) [ref. 1], "IRIX Checkpoint and Restart (CPR) is a facility for saving a running process or set of processes and, at some later time, restarting the saved process or processes from the point already reached, without starting all over again. The checkpoint image is saved in a set of disk files, and restarted by reading saved state from these files to resume execution. Processes can continue to run after checkpoint, and can be checkpointed multiple times."

The cpr utility can be used either through a command-line interface or a graphical user interface called cview. The use of cview was not tested and thus is not included in this paper. The command-line interface includes options to (i) checkpoint, (ii) query a statefile, (iii) restart, and (iv) delete a statefile. The synopses of these options are listed in **Table 1**. When a job is checkpointed, a statefile is created. A statefile is a directory containing all available information about a running process or set of processes (including the names of open files and system objects) to enable restart. The new process(es) should behave just as if the old process(es) had continued.

Table 1. Command-line options of cpr

checkpoint	cpr -c statefile -p id[:type],[id[:type]...] [-fgku]
query	cpr -i statefile
restart	cpr [-j] -r statefile
delete	cpr -D statefile

Table 2. Process(es) types for cpr

PID	for Unix process and POSIX pthread ID (the default type) (use 'ps' command to find PID)
HID	for process hierarchy (tree) rooted at that PID
GID	for Unix process group ID
SID	for Unix process session ID; see termio(7)
ASH	for IRIX Array Session ID; see array_services(5) (use 'array ps' command to find ASH ID)
SGP	for IRIX sproc shared group; see sproc(2)

The -p option used in conjunction with the -c option for checkpoint specifies the process or set of processes to checkpoint. Processes may have any type listed in **Table 2**. Explanation of the other cpr options can be found in the manpage or the IRIX Checkpoint and Restart Operation Guide [ref. 1].

Only the owner of process(es) and the super-user are permitted to perform checkpoint and restart. The rules are as follows:

1. Only the checkpoint owner or superuser is permitted to perform a checkpoint.
2. If the processes have multiple owners, only the superuser is permitted to checkpoint them.
3. Only the checkpoint owner or superuser can restart checkpointed process(es) from a statefile.
4. If the superuser performed a checkpoint, only the superuser can restart it.

PBS Support for CPR

The Portable Batch System (PBS) is a batch queuing system for distributed job management. It was originally developed at NASA as a replacement for NQE. PBS also supports job checkpoint/restart for the Irix operating system. Description of how to checkpoint/restart through PBS is included in the following section.

Procedures for Checkpoint/Restart

Four different procedures are described here for various circumstances. Different fonts are used to distinguish between text (Times New Roman) and commands (Courier New) used in examples.

1. Use CPR for Interactive Sessions

This method is suitable for jobs running in interactive sessions. Checkpoint/restart a resource-intensive job regularly by the job owner prevents loss of valuable data if a system crashes. The steps for this task are straight-forward as shown below.

(a) First start your job in the background, and a process ID (PID), for example 19432, will be returned. If a parent process spawns child processes, the returned PID is the parent process ID. The process(es) should be running and can be seen using either the ps or top commands.

```
% a.out&
[1] 19432
```

(b) Checkpoint a job later and save the information to a statefile. Use a systematic way for naming the statefiles such as including date as part of the names or numbering your statefiles consecutively. In the example shown here, process type HID was used and the processes were killed after they were checkpointed. The chk1 directory should be created in the working directory and it was -rwx- only by root.

```
% cpr -c chk1 -p 19432:HID -k (&)
Checkpointing id 19432 (type HID) to directory chk1
CPR Notice(19469): CPR attrifile /u/db/schang/.cpr is not found
Checkpoint done
```

(c) Restart a job is simple. If a job is successfully restarted, its process(es) should be visible using the ps or top commands.

```
% cpr -r chk1 (&)
Restarting processes from directory chk1
CPR Notice(19458): CPR attrifile /u/db/schang/.cpr is not found
Process restarted successfully
```

(d) Checkpoin/restart can be repeated many times as shown below:

```
% cpr -c chk1 -p 19432:HID -k (&)
% cpr -r chk1 (&)
% cpr -c chk2 -p 19432:HID -k (&)
% cpr -r chk2 (&)
% cpr -c chk3 -p 19432:HID -k (&)
```

(e) An earlier checkpointed state, for example, chk1, can be restarted again even if it has been restarted already. In order for this to succeed, all necessary files associated with that checkpointed state should be saved properly.

```
% cpr -c chk1 -p 19432:HID -k (&)
% cpr -r chk1 (&)
% cpr -c chk2 -p 19432:HID -k (&)
% cpr -r chk1 (&)
```

2. Use CPR in PBS Scripts for Batch Jobs

This method is suitable for jobs running in batch mode. The job owner can use this method for checkpoint/restart if his/her job needs a very long wall-time which exceeds the limit of any batch queues in the system. If the wall-time of the job needed is within the limit of the batch queues, the two methods described later can be used.

The steps used in this second method are very similar to those described in the first method except that cpr is invoked within a PBS script.

(a) Prepare a PBS script (PBS_script1 shown in **Table 3**) that starts the process(es) and then invokes cpr for the first checkpointing. Since a PID is required in order to checkpoint a job, ps, grep and awk commands are used in PBS_script1 to extract the PID.

(b) Submit PBS_script1 using qsub and the system returns with a Job ID, for example 101. After the job has been successfully checkpointed, the statefile chk1 will be created in the working directory. The PBS stdout and stderr files associated with this job (101.fermi) will be moved from /PBS/spool to the user's working directory.

```
% qsub PBS_script1
101.fermi.nas.nasa.gov
```

(c) For subsequent restart and checkpoint cycles, find the process ID, for example 19432, from the PBS stdout associated with PBS_script1 and then prepare PBS_script2 as in **Table 3**.

(d) Submit PBS_script2 and the system returns with a new Job ID, for example 102.

```
% qsub PBS_script2
102.fermi.nas.nasa.gov
```

Table 3. PBS scripts and Gaussian script used in this paper.

PBS_script1

```
#PBS -l ncpus=2
#PBS -l mem=50mw
#PBS -l walltime=1:00:00
cd $PBS_O_WORKDIR
a.out &
sleep 20
cpr -c chk1 -p `ps -u userid | grep a.out | awk '{print $1}'`:HID -k
```

PBS_script2

```
#PBS -l ncpus=2
#PBS -l mem=50mw
#PBS -l walltime=1:00:00
cd $PBS_O_WORKDIR
cpr -r chk1&
sleep 60
cpr -c chk2 -p 19432:HID -k
```

PBS_script3

```
#PBS -l ncpus=2
#PBS -l mem=50mw
#PBS -l walltime=1:00:00
cd $PBS_O_WORKDIR
a.out
```

PBS_script4

```
#PBS -l ncpus=2
#PBS -l mem=50mw
#PBS -l walltime=1:00:00
#PBS -c c=3
cd $PBS_O_WORKDIR
a.out
```

O2.com

```
%nproc=2
%chk=o2.chk
#p ccsd/6-31g* OPT

Gaussian Test O2 molecule

0 1
O
O 1 1.50
```

3. Use PBS qhold and qrls Commands for Batch Jobs

This method is suitable if a system is to be shutdown by a system administrator and all jobs are to be checkpointed before system shutdown and restarted after system recovery. It is also useful if a system administrator wants to improve a system's load balancing and scheduling by holding one or more running jobs. A job owner can also hold and release his/her own jobs. In the sample script shown in **Table 3**, PBS_script3, no cpr is invoked directly. Instead, cpr routines are called by PBS source code when users or system administrators issue the qhold and/or qrls commands.

(a) Prepare a PBS script, PBS_script3 as in **Table 3**.

(b) Submit the script to PBS and the system returns with a Job ID, for example 103.

```
%qsub PBS_script3
103.fermi.nas.nasa.gov
```

(c) If this job is running and a qhold is issued, the job will be checkpointed and a statefile (103.fermi.CK) will be created under /PBS/checkpoint. The job will then be terminated. The job status as seen by using the qstat command should be changed from 'R' to 'H'.

```
%qhold 103
%ls /PBS/checkpoint
103.fermi.CK
```

(d) When a job is to be restarted, qrls is issued. The job status will first be changed from 'H' to 'Q' and later from 'Q' to 'R' when the system is ready to run this job again.

```
%qrls 103
```

(e) If the job runs to completion, the statefile will be deleted. If qhold is issued before the job is completed, the statefile 103.fermi.CK will be updated again and the job status will be changed from 'R' to 'H'.

4. Use PBS -c Option for Auto-checkpointing Batch Jobs Periodically

This last method is suitable if a job owner wishes to safe-guard his/her jobs from system failures by checkpointing it periodically through PBS without directly invoking the cpr command. The PBS -c option, as demonstrated in sample script PBS_script4 in **Table 3**, allows a user to specify the interval in minutes that a job is to be checkpointed periodically. In the example shown in PBS_script4, this interval is set to be 3 minutes. Alternatively, one can also specify the time interval for periodic checkpointing in the qsub command line. The statefile, for

example 104.fermi.CK, will be created in /PBS/checkpoint and updated every 3 minutes. If a job runs to completion, the statefile (104.fermi.CK) created will be deleted. If a job is killed before completion due to system failures, the statefile will remain in /PBS/checkpoint. When the system comes back up, PBS shall automatically restart a job which has a statefile associated with it.

```
%qsub PBS_script4 (or qsub -c c=3 PBS_script3)
%104.fermi.nas.nasa.gov
%ls /PBS/checkpoint
104.fermi.CK
```

Applications Used for Testing

Two types of applications were tested including a simple Gaussian job and an MPI job.

1. A Gaussian Job

The Quantum Chemistry package Gaussian allows certain types of jobs [ref. 2] to be checkpointed and restarted at certain stages within Gaussian by using the 'Restart' key word in a Gaussian job script. However, other types of Gaussian jobs (for example, CCSD, MP4, etc) cannot be checkpointed and restarted within Gaussian. Typically, these Gaussian jobs use a lot of resources and run for a very long time. For them, checkpoint and restart have to be performed through the operating system.

A Gaussian job, defined in the Gaussian script o2.com in **Table 3**, which performed geometry optimization for O2 molecule using the 6-31G* basis set and the CCSD level of method was used to test the four different procedures described earlier for checkpoint/restart by the operating system. The following command was used in place of a.out in the PBS scripts in **Table 3** to start the Gaussian process(es).

```
g98 o2.com o2.out
```

2. An MPI Job

In order to checkpoint an MPI job, the processes associated with it must be run on a single host. In addition, either -cpr or -miser option has to be used with the 'mpirun' command. This is due to the fact that SGI's checkpoint system call cannot checkpoint processes that have open sockets. Therefore, it is necessary to tell mpirun to either not create open sockets or to close an open socket to the array services daemon used to initiate the parallel processes. The -cpr option directs mpirun to close its connection to the array services daemon when a checkpoint is to occur. The -miser option directs mpirun to directly create the parallel process rather than use the array services. This avoids opening the socket connection altogether. More description of these two options can be found from the mpirun manpage.

A simple MPI program that calculates the value of π is used for testing checkpoint/restart using the four different procedures described earlier. Calculation of the value of π is repeated many times. Each time, a different interval h is used to get different accuracy for π. In order for this job to be checkpointed/restarted, -cpr or -miser option is used to start the processes as shown in the following commands which take the place of a.out in the PBS scripts in **Table 3.**

```
mpirun -miser -np 3 ./pi > pi.out
mpirun -cpr -np 3 ./pi > pi.out
```

Results

Table 4 lists the NAS Origin 2000 systems used for the testing of checkpoint/restart described above. For ease of discussion, each system is labeled A, B,…E', respectively as shown in the first column of this table. Both production systems including turing, hopper and lomax and test-bed systems including t3 and evelyn were used. For each machine, the total number of cpus, the version of Irix OS [ref. 3], and the version of PBS on the system at the time tests were performed are listed in columns 3-5. For the production machines at NAS, as shown in columns 6-8, they were configured with (i) a common front-end machine, turing, for running interactive jobs and for batch job submission, (ii) a common PBS server machine, fermi, that accepts batch jobs submitted from turing and coordinates with the PBS scheduler and mom located in the various running hosts. For the test-bed systems, t3 and evelyn, the front-end, the PBS server, PBS scheduler and PBS mom were all in the same host.

Table 4. NAS machines used for testing checkpoint/restart

Label	Host	ncpus	OS	PBS	Front-end	Server	Sched/Mom	year	1	2	3	4
A	turing	24	6.5.7f	N/A	turing	N/A	N/A	2000	x			
B	hopper	64	6.5.4m	2.1p11	turing	fermi	hopper	2000	x	x		
C	lomax	512	6.5.8m	2.3p5	turing	fermi	lomax	2000	x	x		
A'	turing	24	6.5.10f	N/A	turing	N/A	N/A	2001	x			
B'	hopper	64	6.5.10f	Pro_5.0.6	turing	fermi	hopper	2001	x	x	x	x
C'	lomax	512	6.5.10f	Pro_5.0.6	turing	fermi	lomax	2001	x	x		
D	evelyn	8	6.5.7m	Pro_5.0.1	evelyn	evelyn	evelyn	2001	x	x	x	x
E	t3	4	6.5.10f	Pro_5.0.1	t3	t3	t3	2000	x	x		
E'	t3	4	6.5.11f	Pro_5.0.6	t3	t3	t3	2001	x	x	x	x

Column 9 shows when the tests were performed. For example, for the production machines, turing, hopper and lomax, tests were performed both in the second half of year 2000 (systems A-C) before system upgrade and in May of year 2001 (systems A'-C') after system upgrade, respectively. Each of the last four columns (columns 10-13) of **Table 4** shows if a checkpoint/restart procedure was tested at the time the tests were performed. The titles 1, 2, 3, and 4 for these four columns are associated with (i) using cpr interactively, (ii) using cpr in PBS

scripts, (iii) using the PBS qhold/qrls commands, and (iv) using PBS -c option for checkpoint/restart. For example, an x under the column with title 1 means that procedure 1 which used cpr in interactive session was tested.

The results presented here do not include testing of job restarts after system shutdown and system crash. Such tests will be performed in the near future.

As will be seen later, our results show checkpoint/restart are generally more successful on the test-bed systems than on the production systems. Only one failure was encountered on the test-bed system, t3, while using the PBS's qhold/qrls commands for job checkpoint/restart. The causes of this trend are still unknown. The way that the production systems at NAS are configured, with a common front-end, various xfs and nfs mounted file systems, and PBS server, scheduler and mom running on different hosts, etc, might have caused checkpoint/restart to fail easier on these systems than on the test-bed systems. Collaborative efforts from SGI's system engineers, PBS developers and our system administrators are needed for thorough investigation.

In this paper, two caveats and five different types of failures encountered while testing checkpoint/restart are documented. For each of the failures, the systems used and the error encountered, if traceable, are listed.

Caveat 1: Checkpoint/Restart a Multiple-Processor Gaussian Job

The caveat described here applies to all systems (A-E') listed in **Table 4**. Specifically, a multiple-processor Gaussian job, as in the test case o2.com in **Table 3** which runs with 2 processors, does not automatically clear its 'shared-memory segments' when the job is aborted abnormally or checkpointed. Un-cleared shared-memory segments cause memory leak in the system and affect jobs that run afterwards. Although the un-cleared shared memory segments did not in general cause checkpoint/restart to fail on the NAS systems listed in **Table 4**, it was known to cause checkpoint/restart failure on systems running with earlier versions of Irix operating system [ref. 4]. It is thus a good practice for a user to always clear these shared-memory segments after a checkpoint is performed. The ipcs and ipcrm commands can be used for clearing these segments. A script called clearipc which invokes ipcs and ipcrm is also available in the Gaussian 98 package for clearing these shared-memory segments.

```
% ipcs -a
Shared Memory:
T         ID          OWNER
m         62         userid
% ipcrm -m 62
```

Caveat 2: The PBS stdout and stderr Files

This second caveat applies to all but systems A and A' listed in **Table 4** since A and A' are not configured to execute batch jobs. Specifically, for the second procedure that uses cpr in

PBS scripts for checkpoint/restart, cpr requires the PBS stdout and/or stderr associated with (and only with) PBS_script1 to be present in order for the first and all subsequent restarts to succeed. Otherwise, an error shown below is encountered while trying to restart:

```
CPR Error: ckpt_fstat: open /PBS/spool/101.fermi.nas.ER (No such file or directory)
```

This behavior is awkward because PBS automatically transfers the stdout and stderr to the user's working directory and removes the copies under /PBS/spool when a PBS job is completed. To avoid this error, a user can simply recreate (by using the touch command) a stdout and stderr under /PBS/spool.

```
% touch /PBS/spool/101.fermi.nas.ER
% touch /PBS/spool/101.fermi.nas.OU
```

An alternative to avoid this trouble is to start the process(es) and do the first checkpoint through an interactive session instead of a batch session. For systems that do not allow login directly, such as systems B, B', C and C', an interactive PBS session using 'qsub -I' can be used for this purpose. One can restart and do more checkpoint/restart through batch mode from then on without worrying about the presence of the stdout and stderr files for PBS_script1 in /PBS/spool at all.

Failure 1: Checkpoint Fails for an Interactive MPI Job

This failure was seen on systems A, B, C, A', B', and C' for the test MPI job. Specifically, if the MPI job writes output to a file (for example, pi.out), checkpoint fails on A, B, C, A', B' and C' but succeeds on the test-bed systems. If the output is not written to a file, checkpoint succeeds on all systems. This behavior is demonstrated below:

Checkpoint fails on A, B, C, A', B' and C' but not on test-bed systems if the job writes output to a file (in an xfs file system):

```
% mpirun -miser -np 3 ./pi > pi.out &
[ 1] xxxxx
% cpr -c chk1 -p xxxxx:HID -k
Checkpointing id xxxxx (type HID) to directory chk1
(….. no progress made ever, checkpoint stalls)
```

Checkpoint is successful on all systems if no output file is used as shown here:

```
% mpirun -miser -np 3 ./pi &
[ 1] xxxxx
```

```
% cpr -c chk1 -p xxxxx:HID -k
Checkpointing id xxxxx (type HID) to directory chk1
Checkpoint done
```

Failure 2: Restart Fails After a Few Successful Cycles of Checkpoint/Restart Through PBS

This failure was seen when cpr was used in PBS scripts and applies to both the Gaussian and the MPI test jobs. It was seen on systems B, C, B', and C' but not on the test-bed systems. Also, this failure does not apply to systems A and A' since batch jobs are not executed on them. Specifically, checkpoint/restart through PBS is successful for a few cycles. But it fails to do so in a later cycle.

```
cpr -c chk1 -p xxxxx:HID -k     (successful)
cpr -r chk1                     (successful)
cpr -c chk2 -p xxxxx:HID -k     (successful)
cpr -r chk2                     (successful)
….
cpr -c chkn -p xxxxx:HID -k     (successful)
cpr -r chkn                     (failed)
```

The PBS stdout and stderr show the error complained by cpr as follows:

```
Restart chkn failed
….
CPR Error: Failed to place mld 0 (Invalid argument)
CPR Error: Unexpected status EOF
CPR Error: Cleaning up the failed restart
```

Failure 3: Restart Fails From a Checkpointed State Which Was Once Restarted Successfully

This is a continuation of the previous example that shows cpr is successful for a few cycles but it fails in a later cycle. Under this circumstance, one would expect that going back to an earlier checkpoint state which has been restarted successfully once and repeating restart from that state should succeed. In this example, a failure occurs when trying to do so. For example, restarting from the 'chk2' state was once successful but it failed in a later attempt.

```
cpr -c chk1 -p xxxxx:HID -k     (successful)
cpr -r chk1                     (successful)
cpr -c chk2 -p xxxxx:HID -k     (successful)
cpr -r chk2                     (successful)
….
```

```
cpr -c chkn -p xxxxx:HID -k     (successful)
cpr -r chkn                     (failed)
cpr -r chk2                     (once successfully, but failed later)
```

The error messages from PBS stdout and stderr are the same as those seen in the previous failure described above.

Failure 4: Restart Fails Using qhold/qrls Due to Cpusets Problem

This type of failure was seen in systems B' when job checkpoint/restart was performed using the PBS qhold and qrls commands. It applies to both the Gaussian and the MPI jobs.

```
turing% qsub PBS_script3
8128.fermi.nas.nasa.gov
turing% qhold 8128    (successful, job held, processes stopped, 8128.fermi.CK created)
turing% qrls 8128     (successful, job restarted, processes resumed)
turing% qhold 8128    (successful, job held, processes stopped, 8128.fermi.CK updated)
turing% qrls 8128     (failed)
```

As shown below, job 8128 and its processes were not restarted because the kernel thought that a file called cpuset.8128.fer already existed and thus cpuset.8128.fer could not be created for the request to restart.

```
pbs_mom;File exists (17) in create_cpuset, failed to create cpuset 8128.fer
pbs_mom;Svr;pbs_mom;File exists (17) in mach_restart, Cannot assign cpuset to
8128.fermi.nas.nasa.gov
pbs_mom;Svr;pbs_mom;Error 0 (0) in mom_restart_job, 8128.fermi.nas.nasa.gov:
task 1 failed from file /PBS/checkpoint/8128.fermi..CK/0000000001
pbs_mom;Job;8128.fermi.nas.nasa.gov;Restart failed, error 0
pbs_mom;Job;8128.fermi.nas.nasa.gov;kill_job
```

Another anomaly is that when qrls failed to restart the process(es), PBS also failed to realize that the job was not restarted successfully and still listed the job as running ('R' in qstat). Furthermore, this job could not be killed using the regular 'qdel' command. Instead, a system administrator had to clean up such jobs manually.

Failure 5: Restart Fails Using qhold/qrls Due to PBS Time-Tracking Error

This failure was seen in the test-bed system E' when job checkpoint/restart was performed using the PBS qhold and qrls commands. It was encountered while testing with the MPI job. Specifically, when a job was restarted using the qrls command, the processes appeared to be restarted properly and ran for some time but they were killed abnormally as shown below:

```
turing% qsub PBS_script3
148.t3.nas.nasa.gov
turing% qhold 148        (successful, job held, processes stopped, 148.t3.CK created)
turing% qrls 148         (successful, job restarted, processes resumed and ran for a
                          short period of time and then were killed abnormally)
```

In addition, after the processes were killed, PBS failed to recognize that this job should be removed from PBS. Instead, this job was stilled listed as running ('R' in qstat) and could not be deleted with a simple qdel command. This behavior is the same as the failure just described in the previous section for the failure encountered on system B'.

From the PBS log file, it showed that the job was restarted and ran for 40 seconds and the processes got killed because walltime was exceeded. Further investigation showed that this was simply a time-tracking error in the PBS mom source code. The bug causing this error was identified and reported to the developers of PBS. A fix has been supplied for this bug.

```
05/07/2001 11:17:04  S    Job Queued at request of schang
05/07/2001 11:17:05  M    Started, pid = 27864
05/07/2001 11:17:37  A    05/07/2001 11:17:37  S Holds u set at request of
schang

05/07/2001 11:17:39  M    checkpointed to /PBS/checkpoint/148.t3.nas..CK
05/07/2001 11:18:51  S    Holds u released at request of schang
…..
05/07/2001 11:19:01  M    adjust wall start to Mon May  7 11:21:31 2001
05/07/2001 11:19:01  M    Restarted 1 tasks
05/07/2001 11:19:41  M    walltime 00:00:-110 exceeded limit 00:15:00
05/07/2001 11:19:41  M    kill_job
```

Conclusion

Four different procedures for performing checkpoint/restart using the Irix utility cpr and PBS are described and tested for two applications: a Gaussian job and an MPI job. Two caveats and five different types of failures are reported in this work. Failures were encountered more often on our production systems than the test-bed systems. The causes of this trend are not clear and further investigations are needed. Nevertheless, the results presented show that checkpoint/restart is complicated and is still not reliable for the NAS systems.

The tests performed so far are by no means extensive and did not include all possible scenarios. More tests are needed to uncover possible bugs in the operating system, cpr and/or PBS. Specifically, future testing will include: (i) a wide variety of user applications such as OpenMP, pvm, mpi programs, etc. (ii) large parallel jobs, (iii) system-wide checkpoint/restart,

(iv) system crash simulation, and (v) efficiency. In addition, we need to have more communication among (i) SGI engineers, (ii) PBS developers, and (iii) system administrators to discuss problems and bug-fixes in order to monitor the progress on the various checkpoint/restart issues.

Our ultimate goal is to make sure checkpoint/restart is reliable in a real production environment. Until this is possible, users are still encouraged to engineer checkpoint/restart in their own programs and scripts in order to safe-guard their valuable data.

Acknowledgement

Assistance from Ed Hook, Lorraine Freeman, Chuck Niggley, the SGI on-site analyst Bron Nelson and support from the NASA Advanced Supercomputing Division under contract number ARC330.000.2 are greatly appreciated.

References

1. IRIX Checkpoint and Restart Operation Guide 007-3236-004, Silicon Graphics, Inc.
2. Gaussian 98 User's Reference, Revision A. 1, Gaussian, Inc.
3. The OS patches included for the systems are : system A' (turing), 3965, 4116, 4123, 4156, 4180; system B' (hopper), 3965, 4116, 4123, 4156, 4173, 4180; system C' (lomax), 3965, 4097, 4116, 4123; and system E' (t3), 4193, 4195, 4209, 4242, and 4258.
4. Private communication with Dr. Joseph Ochterski of Gaussian Inc.

   "CPR Error: sys V shared memory restore failed for shmid: 30: id in use: Pid : 9286"