An Overview of Using Condor with DMTCP Checkpointing

Samaneh Kazemi

Version 1.0 (July 3, 2012)

Contents

NEWS (July 3, 2012): *There appears to be a bug related to the dependency-based booting of Debian 6.0/squeeze, which affects the use of Condor/DMTCP. If you use Debian squeeze, please see* `https://condor-wiki.cs.wisc.edu/index.cgi/tktview?tn=3094` .

1 Starting Point

DMTCP (Distributed MultiThreaded CheckPointing) can be found at http://dmtcp.sourceforge.net. The primary web page for using DMTCP checkpointing with the Condor Vanilla Universe can be found at https://condor-wiki.cs.wisc.edu/index.cgi/wiki?p=DmtcpCondor under the name dmtcp_condor_integration.

For those who are already familar with Condor, it suffices to skip immediately to the Appendix A, and copy a version of the submit description file found there. You will also need the `shim_dmtcp` script, found on the Condor project pages, or at `http://dmtcp.sourceforge.net/condor.html`. The submit description file runs `shim_dmtcp`, and the arguments to `shim_dmtcp` specify the actual application that you will run in Condor.

For those who would like a gentler intorduction, one uses the `condor_submit` command to submit a job. Its argument is a *submit description file* in which Condor finds everything it needs to know about the job. One specifies parameters in the file, known as *attributes*. The second most important Condor command is `condor_q`. In the header of the output, note especially 'ID' (CLUSTER_ID.JOB_ID) and 'ST' (STATUS). The CLUSTER_ID.JOB_ID (for example, 5690.0) is the name of your job, and is needed for commands such as `condor_ssh_to_job`, `condor_rm`, and `condor_vacate`. The status is usually 'R' (running), 'I' (idle), 'H' (hold), 'C' (completed), 'X' (removed), or '>' (transferring output).

Condor works hard to keep a job running. When a machine must vacate a job from a machine (due to a higher priority use of that machine, time limit, etc.). Condor must kill or migrate that job. DMTCP (Distributed MultiThreaded CheckPointing) is a checkpointing package that allows jobs to be migrated, or later restarted.

Integrating DMTCP with Condor. In integrating DMTCP with Condor's vanilla universe, it's important to note that Condor doesn't know anything about DMTCP. The user submits a Condor job called `shim_dmtcp`. The program `shim_dmtcp` is a shell script that takes arguments. A typical calling sequence would be:

```
shim_dmtcp --log shim_dmtcp.log --stdout foo.out --stderr foo.err --ckptint 1800 \
  ./foo arg1 ...
```

As a consequence, the Condor submit description file will list `shim_dmtcp` as the Condor job executable. The Condor submit description file will describe `foo` via the Condor attribute, `transfer_input_files`. (I.e., `foo` is an input file for `shim_dmtcp`.)

When Condor vacates a job, it will first send `shim_dmtcp` a soft kill signal (SIGINT). The hard kill signal (SIGQUIT) is sent later in case `shim_dmtcp` does not exit. Normally, `shim_dmtcp` will trap the SIGINT, and call DMTCP to create a checkpoint. Then, `shim_dmtcp` exits.

After killing or vacating a job in the Condor vanilla universe, Condor will assume that the job must be re-started from the beginning on a new host. However, when Condor re-starts `shim_dmtcp`, the shim script will look for checkpoint files in the local directory. If it finds them, it assumes that it should restart an old job, and calls on DMTCP to do so.

2 The Condor Universes

There are several runtime environments in Condor called *universes*. The most common two are the *Standard Universe* and *Vanilla Universe*. The Standard Universe allows you to submit a restricted class of jobs (single-threaded, single-process, etc.) to be run by Condor. Condor's native checkpointer allows jobs to be migrated or later restarted. In contrast, the Vanilla Universe does not support built-in checkpointing.

In the Standard Universe, it is required to compile your program with the Condor libraries. For additional restrictions, see:
http://research.cs.wisc.edu/condor/manual/v7.6/2_4Road_map_Running.html#sec:standard-univ-erse

3 Condor's File Transfer Mechanism

Internally, Condor copies the user's files to a *sandbox directory* (sometimes known as a "build" or "working" directory) on the exec host. Sandbox directories are created as needed. The location of the sandbox directory is specified by the *LOCAL_DIR* variable of the configuration file. The location of the Condor configuration file varies. The recommended way to find the location is issuing the "`condor_config_val -config`" command. A convenient alternative is the command "`condor_config_val EXECUTE`".

Condor executes the job on the exec host, and transfers output back to the submit directory of the submit host. The submit description file specifies which files and directories to transfer, and at what point the output files should be copied back to the submitting machine. If the job does not finish, for instance because of an error during execution, the files will not be transferred back to the submit host. (See Section 6 for further information on submit description files.)

4 Basic Condor Commands

`condor_submit job.sub`: Queue jobs for execution on remote machines. This requires the submit description file, `job.sub`. See Section 6 for details on Condor submit description files.

`condor_q`: Displays information(such as owner or ID) about all the jobs in queue. See Section 5 for a list of the Condor jobs stages. One might use options such as `-global`, `-submitter`, `-run` or `-name`:
http://research.cs.wisc.edu/condor/manual/v6.2/condor_q.html

`condor_status -state -total`: List a summary of pool resources.

`condor_rm ID` or `condor_rm all`: Remove a specific job or all of the jobs from queue.

`condor_vacate_job` (of interest for debugging DMTCP under Condor): This finds one or more jobs from the Condor job queue and vacates them from the host(s) where they are currently running. (Condor may also spontaneously vacate a job for lack of resources.) The vacated jobs remain in the job queue and return to the idle state. (This usually happens fast, but if a DMTCP checkpoint takes longer, then this may take some time.) One can specify the job by its ID or by its owner. A job that is vacated will be sent a *soft kill signal*, typically SIGINT. Condor will then restart the job from the beginning on a new exec host. Using `condor_vacate_job` on jobs which are not currently running has no effect.

There is special handling for a job in the Standard Universe, or a job using `shim_dmtcp` in the Vanilla Universe. In these cases, the job will first produce a checkpoint, and then the job will be killed. Condor will then restart the job on a new exec host, using the checkpoint to continue from where it left off.

`condor_ssh_to_job` (of interest for debugging DMTCP under Condor): This is available on more recent versions of Condor. It will ssh to the exec host, and cd to the working directory of your job *provided your job is still running.*

5 Status of a job on a Condor exec host

The `condor_q` command indicates the state of Condor jobs. Some of the status states are:

- **H:** on hold (In the hold state, the job will not be scheduled to run until it is released. See the man pages for `condor_hold` and `condor_release`.)
- **R:** running
- **I:** idle (Waiting for a machine on which to execute. Note that `condor_q -better-analyze` will report if both the job is idle and the job fails to match the required resources specified in the submit description file.)
- **C:** completed
- **X:** removed

Descriptions of some of the state transitions follow.

R → H: When a problem occurs during the execution such as a missing file that was supposed to be transferred, Condor will hold the job. Held jobs remain in the queue, waiting for user intervention.

H→R: Upon resolving the problem above, the command `condor_release` frees the job for continued execution.

R → I → R: If one vacates a job, Condor stops running the job and the job status goes to idle. After vacating a job, Condor will look for an available machine, on which to restart the job. Note that if a checkpoint is being used, it will restart from the last checkpoint image. Otherwise, it will restart from the beginning of the program. This process often takes minutes.

6 Sample Submit Description Files

The `condor_submit` command requires a condor submit description filename as a parameter. There are three basic examples to start using Condor in:
http://research.cs.wisc.edu/condor/manual/v7.6.2/2_5Submitting_Job.html#SECTION00351000000000000000

6.1 Transfering Input Files

The executable (`a.out` in this example) and any other input files must be specified by an attribute in the Condor submit description file. All filenames are relative to the submit directory.

```
transfer_input_files = a.out
```

The attribute above depends on DMTCP being installed site-wide. If DMTCP is not installed site-wide, then the user must provide his or her own copy of all DMTCP executables and libraries as part of the same Condor attribute.

```
transfer_input_files = dmtcp_checkpoint,dmtcp_coordinator,

dmtcp_command,dmtcp_restart,dmtcphijack.so,libmtcp.so,mtcp_restart,a.out
```

6.2 Transfering Output Files

The shim_dmtcp script will save the standard output and standard error of the job (e.g. a.out) in the working directory of the exec host. In order to transfer this, and any files written by a.out, one must include the following attributes in the submit description file:

```
should_transfer_files = YES
when_to_transfer_output = ON_EXIT_OR_EVICT
```

Note also that if the job does not finish, for instance because of an error during execution, the output files will not be transferred back to the submit host. (Optionally, if Condor parameters `output`, `error`, and `log` are specified in the Condor description file, then the specified files will be created on output in the user's submit directory. However, we have not found these useful in conjunction with the DMTCP shim script for Condor.)

7 The shim_dmtcp Script

In order to use the DMTCP checkpointing package, some additional work is needed before submitting the desired job. A `shim_dmtcp` file is provided for this purpose. This section describes both the use and some internals of the `shim_dmtcp` script.

7.1 Usage of shim_dmtcp Script

One needs to pass the shim_dmtcp file to *executable* and the name of the job (for example `foo.o`) to *arguments* in the submit description file.

```
executable = shim_dmtcp

arguments = --log shim_dmtcp.$(CLUSTER).$(PROCESS).log --stdin foo.c --stdout
    job.$(CLUSTER).$(PROCESS).out --stderr job.$(CLUSTER).$(PROCESS).err
    --ckptint 1800 ./foo 7200 output_file.$(CLUSTER).$(PROCESS).out
```

7.2 Preparing the DMTCP files prior to the Condor job

Assume that:

```
DMTCP_BIN=full pathanme for dmtcp/bin
```

```
DMTCP_LIB=full pathname for dmtcp/lib
```

Then the following files must be copied to the submit directory (using a flat file structure):

```
$(DMTCP_BIN)/dmtcp_checkpoint
$(DMTCP_BIN)/dmtcp_command
$(DMTCP_BIN)/dmtcp_coordinator
$(DMTCP_BIN)/dmtcp_restart
$(DMTCP_LIB)/dmtcp/dmtcphijack.so
$(DMTCP_LIB)/libmtcp.so
$(DMTCP_LIB)/libmtcp.so.1
$(DMTCP_BIN)/mtcp_restart
```

8 Debugging Condor/DMTCP jobs and Internals of shim_dmtcp

Before submitting the shim_dmtcp file, the DMTCP files must be prepared in the submit directory.

Once your Condor job is running, if you have the command `condor_ssh_to_job`, you can use it to ssh to the sandbox directory on the exec host. If not, you can alternatively find the exec host via: `condor_q -long | grep RemoteHost`
On the exec host, use the Linux `ps` command to find your process. `ps -elf` will show the full pathname of your executable (renamed to condor_*.exe). The sandbox directory will typically be of the form: `/var/lib/condor/execute/dir_*`

After running a job, one can diagnose the results using DMTCP log files. If the job fails, one must debug what went wrong. This section describes these issues.

8.1 Internals of shim_dmtcp Script

When `shim_dmtcp` begins, it will execute the followings:

1) Close the GCB file descriptor.
   Condor's Linux releases (version 6.9.5 and later under the Apache Licence version 2.0.) are linked with GCB, Generic Connection Brokering. GCB is a firewall/NAT traversal solution that enables communications through the firewall/NAT. It consists of daemon processes. The file descriptor inherited by GCB is closed by `shim_dmtcp` before any `dmtcp_checkpoint` is called, because external socket cannot be restored by DMTCP.

2) Run `dmtcp_coordinator` using the port and checkpoint interval time specified in the submit description file. It is run on the same remote host machine of Condor as `shim_dmtcp`:
   `./dmtcp_coordinator --port $port --exit-on-last --interval $INTERVAL --background`

3) Run `dmtcp_checkpoint` by specifying the port for the given job. For instance, to execute "./fib 2", `shim_dmtcp` calls the following:
   `dmtcp_checkpoint --port $port -j ./fib 2`

4) The `shim_dmtcp` script creates a trap to handle SIGINT (Condor's signal for vacating a job). When the SIGINT is received, the following commands are executed:
   `dmtcp_command --port $port --checkpoint`

`dmtcp_command --port $port --status`
`dmtcp_command --port $port --kill`
`./dmtcp_restart_script.sh` (in case it exists)

8.2 Debugging a Condor Job Using DMTCP

1. *How to inspect the results during a DMTCP job:*
   The command `condor_ssh_to_job` allows one to ssh to the exec host in the current directory of the job. The DMTCP `jassertlog.*` files will be there, and other standard DMTCP debugging techniques can be used.

2. *How to inspect the results after a DMTCP job:*
   Make sure that `DMTCP_DIR=./` is part of the environment parameter of the Condor submit description file. After a DMTCP job, three `jassertlog.*_*` files will be created (jassertlog.*_dmtcp_coordinator, jassertlog.*_dmtcp_checkpoint, and jassertlog.*_foo) in the current directory. If the keyword ERROR does not appear in any of these files, then the DMTCP job succeeded. In the case of success, the stdout for the user application will appear in file specified as the value of `--stdout` for the `arguments` parameter of the shim script. In the example in Section A, the value of `--stdout` is `job.$(CLUSTER).$(PROCESS).out`.

3. *Where are ckpt*.dmtcp files placed?*
   They will be placed in the same directory where the executable exists (sandbox directory on the remote host).

4. If one configures dmtcp with `./configure --enable-debug`, it will create the directory `/tmp/dmtcp-USER@HOST`, which contains debugging files of the form `jassertlog.*_*`.

5. *What do I do to get a copy of* `ckpt_*dmtcp`*,* `jassertlog._*`*, etc?*
   If the submit description file set `when_to_transfer_output`, then Condor will copy newly created files on the exec host to the directory of the submit host where the submit command was called. Condor searches for newly created files in a search path specified by `environment=PATHS`, where `PATHS` is a semicolon-separated sequence of search paths on the exec host. The `environment` is specified in the submit description file. If a path is relative, it is relative to the remote sandbox directory.

   Assume one also wishes to see the DMTCP log files (typically found at `/tmp/dmtcp-USER@HOST`, then one must add this to `environment` in the submit description file. See the example submit description file in Section A.

Acknowledgement

The author thanks Gene Cooperman and Alan De Smet for careful reading and comments on earlier versions.

A Appendix: Condor Submit Description File

.

An example `submit description file` to use with `shim_dmtcp` for the Vanilla Universe follows. The first version assumes that DMTCP is already installed globally at your site. The second version allows you to use DMTCP with Condor from a privately installed DMTCP. Both versions assume that `shim_dmtcp` will execute the actual application, `./foo`. By using `shim_dmtcp` and DMTCP, if Condor evicts a job, then Condor will automatically restart it, continuing execution where it left off.

A.1 Submit Description File: assumes DMTCP installed globally

```
universe = vanilla
executable = shim_dmtcp
coresize=-1

##############################################################################
# Argument Meaning
#------------------
# --log     log file name for actions in shim_dmtcp script, if n/a use /dev/null
# --stdin   stdin file, if n/a use /dev/null
# --stdout  stdout file, if n/a use /dev/null
# --stderr  stderr file, if n/a use /dev/null
# --ckptint checkpointing interval in seconds
# 1         the executable name you should have transferred in
# 2+        arguments to the executable
##############################################################################
arguments = --log shim_dmtcp.$(CLUSTER).$(PROCESS).log --stdout \
 job.$(CLUSTER).$(PROCESS).out --stderr job.$(CLUSTER).$(PROCESS).err \
 --ckptint 1800 ./foo arg1

##############################################################################
# Enable file transfer. Here is where you 'mixin' the user's input and
# output fles along with what is needed for DMTCP. Don't forget to transfer
# the actual executable along.
##############################################################################
should_transfer_files = YES
when_to_transfer_output = ON_EXIT_OR_EVICT
transfer_input_files = foo

##############################################################################
# Set up various environment variables. If you need to specify more, mix them
# in here.
##############################################################################
environment=DMTCP_TMPDIR=./;JALIB_STDERR_PATH=/dev/null;\
DMTCP_PREFIX_ID=$(CLUSTER)_$(PROCESS)
##############################################################################
# SIGINT is our soft checkpointing signal
##############################################################################
kill_sig = 2

##############################################################################
# Output and log files for the shim process which performs the work.
##############################################################################
output = shim.$(CLUSTER).$(PROCESS).out
error = shim.$(CLUSTER).$(PROCESS).err
log = shim.$(CLUSTER).$(PROCESS).log

Notification = Never

queue 1
```

A.2 Submit Description File: assumes privately installed DMTCP

```
universe = vanilla
executable = shim_dmtcp
coresize=-1
######################### ./foo ######################

##############################################################################
# Testing Various Requirements Expressions
# DMTCP sometimes does odd behaviors on slightly different installations
# of Linux. This allows me to select different types of machines to narrow
# down problems.
##############################################################################

# Keep it on the same checkpoint platform and kernel revision. Change to
# whatever you need this to be to limit the run of DMTCP jobs to a specific
# set of platforms if you discover problems.

#Requirements = (CheckpointPlatform == "LINUX INTEL 2.6.x normal 0x40000000" \
# && OSKernelRelease == "2.6.18-128.el5")

##############################################################################
# Argument Meaning
#-----------------
# --log     log file name for actions in shim_dmtcp script, if n/a use /dev/null
# --stdin   stdin file, if n/a use /dev/null
# --stdout  stdout file, if n/a use /dev/null
# --stderr  stderr file, if n/a use /dev/null
# --ckptint checkpointing interval in seconds
# 1         the executable name you should have transferred in
# 2+        arguments to the executable
##############################################################################
arguments = --log shim_dmtcp.$(CLUSTER).$(PROCESS).log --stdout \
 job.$(CLUSTER).$(PROCESS).out --stderr job.$(CLUSTER).$(PROCESS).err \
 --ckptint 1800 ./foo arg1

##############################################################################
# Enable file transfer. Here is where you 'mixin' the user's input and
# output fles along with what is needed for DMTCP. Don't forget to transfer
# the actual executable along.
##############################################################################
should_transfer_files = YES
when_to_transfer_output = ON_EXIT_OR_EVICT
transfer_input_files = dmtcp_checkpoint,dmtcp_coordinator,dmtcp_command,\
dmtcp_restart,dmtcphijack.so,libmtcp.so,libmtcp.so.1,mtcp_restart,foo

##############################################################################
# Set up various environment variables. If you need to specify more, mix them
# in here.
##############################################################################
environment=DMTCP_TMPDIR=./;JALIB_STDERR_PATH=/dev/null;\
DMTCP_PREFIX_ID=$(CLUSTER)_$(PROCESS)
```

```
##########################################################################################
# SIGINT is our soft checkpointing signal
##########################################################################################
kill_sig = 2

##########################################################################################
# Output and log files for the shim process which performs the work.
##########################################################################################
output = shim.$(CLUSTER).$(PROCESS).out
error = shim.$(CLUSTER).$(PROCESS).err
log = shim.$(CLUSTER).$(PROCESS).log

Notification = Never

queue 1
```