Subject: Re: [Ksummit-2010-discuss] checkpoint-restart: naked patch
Posted by Kapil Arya on Wed, 24 Nov 2010 03:50:44 GMT
View Forum Message <> Reply to Message

(Our first comment below actually replies to an earlier post by Oren. It seemed simpler to combine our comments.)

> > d. screen and other full-screen text programs These are not the only
> > examples of difficult interactions with the rest of the world.
>
> This actually never required a userspace "component" with Zap or linux-cr (to
> the best of my knowledge).

We would guess that Zap would not be able to support screen without a user space component. The bug occurs when screen is configured to have a status line at the bottom. We would be interested if you want to try it and let us know the results.

==========

> > > category   linux-cr
> > > userspace
> > > ---------- ----------
> > > PERFORMANCE   has _zero_ runtime overhead   visible overhead due to
> > > syscalls interposition and state tracking even w/o checkpoints;
> >
> > In our experiments so far, the overhead of system calls has been
> > unmeasurable. We never wrap read() or write(), in order to keep overhead
> > low. We also never wrap pthread synchronization primitives such as locks,
> > for the same reason. The other system calls are used much less often, and
> > so the overhead has been too small to measure in our experiments.
>
> Syscall interception will have visible effect on applications that use those
> syscalls. You may not observe overheasd with HPC ones, but do you have
> numbers on server apps ? apps that use fork/clone and pipes extensively ?
> threads benchmarks et ? compare that to aboslute zero overhead of linux-cr.

Its true that we haven't taken serious data on overhead with server apps. Is there a particular server app that you are thinking of as an example? I would expect fork/clone and pipes to be invoked infrequently in the server apps and do not add measurably to CPU time. In most server apps such as MySQL, it is common to maintain a pool of threads for reuse rather than to repeatedly call clone for a new thread. This is done to ensure that the overhead of the clone calls is not significant. I would expect a similar policy for fork and pipes.

<snip>

> > > OPERATION   applications run unmodified   to do c/r, needs

> > > 'controller' task (launch and manage _entire_ execution) - point of
> > > failure. restricts how a system is used.
> >
> > We'd like to clarify what may be some misconceptions. The DMTCP controller
> > does not launch or manage any tasks. The DMTCP controller is stateless,
> > and is only there to provide a barrier, namespace server, and single point
> > of contact to relay ckpt/restart commands. Recall that the DMTCP
> > controller handls processes across hosts --- not just on a single host.
>
> The controller is another point of failure. I already pointed that the
> (controlled) application crashes when your controller dies, and you mentioned
> it's a bug that should be fixed. But then there will always be a risk for
> another, and another ... You also mentioned that if the controller dies,
> then the app should contionue to run, but will not be checkpointable anymore
> (IIUC).
>
> The point is, that the controller is another point of failure, and makes the
> execution/checkpoint intrusive. It also adds security and user-management
> issues as you'll need one (or more ?) controller per user (right now, it's
> one for all, no ?). and so on.

Just to clarify, DMTCP uses one coordinator for each checkpointable computation. A single user may be running multiple computations with one coordinator for each computation. We don't actually use the word controller in DMTCP terminology because the coordinator is stateless and so in coordinating but not controlling other processes.

> Plus, because the restarted apps get their virtualized IDs from the
> controller, then they can't now "see" existing/new processes that may get the
> "same" pids (virtualization is not in the kernel).

This appears to be a misconception. The wrappers within the user process maintain the pid-translation table for that process. The translation table is the translation between the original pid given by the kernel and the current pid set by the kernel on restart. This is handled locally and does not involve the coordinator.

In the case of a fork there could be a pid-clash (the original pid generated for a
new process that conflicts with someone else's original pid). However, DMTCP handles this by checking within the fork wrapper for a pid-clash. In the rare case of a pid-clash, the child process exits and the parent forks again. Same applies for clone and any pid clash at restart time.

> > Also, in any computation involving multiple processes, _every_ process
> > of the computation is a point of failure. If any process of the
> > computation dies, then the simple application strategy is to give up
> > and revert to an earlier checkpoint. There are techniques by which an

> > app or DMTCP can recreate certain failed processes. DMTCP doesn't
> > currently recreate a dead controller (no demand for it), but it's not
> > hard to do technically.
>
> The point is that you _add_ a point of failure: you make the "checkpoint"
> operation a possible reason for the application to crash. In contrast, in
> linux-cr the checkpoiint is idempotent - nunharmful because it does not make
> the applications execute. Instead, it merely observes their state.

We were speaking above of the case when the process dies during a computation. We were not referring to checkpoint time.

<snip>

We would like to add our own comment/question. To set the context we quote an earlier post:

OL> Even if it did - the question is not how to deal with "glue"
OL> (you demonstrated quite well how to do that with DMTCP), but
OL> how should teh basic, core c/r functionality work - which is
OL> below, and orthogonal to the "glue".

There seems to be an implicit assumption that it is easy to separate the DMTCP "glue code" from the DMTCP C/R engine as separate modules. DMTCP is modular but it splits the problems into modules along a different line than Linux C/R. We look forward to the joint experiment in which we would try to combine DMTCP with Linux C/R. This will help answer the question in our mind.

In order to explore the issue, let's imagine that we have a successful merge of DMTCP and Linux C/R. The following are some user-space glue issues. It's not obvious to us how the merged software will handle these issues.

1. Sockets -- DMTCP handles all sockets in a common manner through a single module. Sockets are checkpointed independently of whether they are local or remote. In a merger of DMTCP and Linux C/R, what does Linux C/R do when it sees remote sockets? Or should DMTCP take down all remote sockets before checkpointing? If DMTCP has to do this, it would be less efficient than the current design which keeps the remote sockets connections alive during checkpoint.

2. XLib and X11-server -- Consider checkpointing a single X11 app without the X11-server and without VNC. This is something we intend to add to DMTCP in the next few months. We have already mapped out the design in our minds. An X11 application includes the Xlib library. The data of an X11 window is, by default, contained in the X11 library -- not in the X11-server. The application communicates with the X11-server using socket connections, which would be considered a leak by Linux C/R. At restart time, DMTCP will ask the X11-server to create a bare window and then make the appropriate Xlib call to

repaint the window based on the data stored in the Xlib library.
For checkpoint/resume, the window stays up and does not has to be repainted.
How will the combined DMTCP/Linux C/R work? Will DMTCP have to take
down the window prior to Linux C/R and paint a new window at resume time?
Doesn't this add inefficiency?

3. Checkpointing a single process (e.g. a bash shell) talking to an xterm via
a pty -- We assume that from the viewpoint of Linux C/R a pty is a leak since
there is a second process operating the master end of the pty. In this
case we are
guessing that Linux C/R would checkpoint and restart without the gurantees of
reliability. We are guessing that Linux C/R would not save and restore the pty,
instead it would be the responsibility of DMTCP to restore the current settings
of the pty (e.g. packet mode vs. regular mode). Is our understanding correct?
Would this work?

Thanks,
Gene and Kapil

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containe rs