Subject: Re: [Ksummit-2010-discuss] checkpoint-restart: naked patch
Posted by Gene Cooperman on Sun, 21 Nov 2010 08:21:43 GMT
View Forum Message <> Reply to Message

As Kapil and I wrote before, we benefited greatly from having talked with Oren,
and learning some more about the context of the discussion. We were able
to understand better the good technical points that Oren was making.
Since the comparison table below concerns DMTCP, we'd like to
state some additional technical points that could affect the conlusions.

> 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.

> OPTIMIZATIONS many optimizations possible limited, less effective
> only in kernel, for downtime, w/ much larger overhead.
> image size, live-migration

As above, we believe that the overhead while running is negligible. I'm
assuming that image size refers to in-kernel advantages for incremental
checkpointing. This is useful for apps where the modified pages tend
not to dominate. We agree with this point. As an orthogonal point,
by default DMTCP compresses all checkpoint images using gzip on the fly.
This is useful even when most pages are modified between checkpoints.
Still, as Oren writes, Linux C/R could also add a userland component
to compress checkpoint images on the fly.
Next, live migration is a question that we simply haven't thought much
about. If it's important, we could think about what userland approaches might
exist, but we have no near-term plans to tackle live migration.

> 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.
    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.

> PREEMPTIVE      checkpoint at any time, use     processes must be runnable and
>                 auxiliary task to save state;   "collaborate" for checkpoint;
>                 non-intrusive: failure does     long task coordination time
>                 not impact checkpointees.       with many tasks/threads. alters
>                                                 state of checkpointee if fails.
>                                                 e.g. cannot checkpoint when in
>                                                 vfork(), ptrace states, etc.

Our current support of vfork and ptrace has some of the issues that Oren points
out.  One example occurs if a process is in the kernel, and a ptrace state has
changed.  If it was important for some application, we would either have
to think of some "hack", or follow Tejun's alternative suggestion to work
with the developers to add further kernel support.  The kernel developers
on this list can estimate the difficulties of kernel support better than I can.

> COVERAGE        save/restore _all_ task state;  needs new ABI for everything:
>                 identify shared resources; can  expose state, provide means to
>                 extend for new kernel features  restore state (e.g. TCP protocol
>                 easily                          options negotiated with peers)

Currently, the only kernel support used by DMTCP is system calls (wrappers),
/proc/*/fd, /proc/*/maps, /proc/*/cmdline, /proc/*/exe, /proc/*/stat.  (I think
I've named them all now.)  The kernel developers will know better
than us what other kernel state one might want to support for C/R, and what
types of applications would need that.

> RELIABILITY     checkpoint w/ single syscall;   non-atomic, cannot find leaks
>                 atomic operation. guaranteed    to determine restartability
>                 restartability for containers

My understanding is that the guarantees apply for Linux containers, but not
for a tree of processes.  Does this imply that linux-cr would have some
of the same reliability issues as DMTCP for a tree of processes?  (I mean
the question sincerely, and am not intending to be rude.)  In any case,
won't DMTCP and Linux C/R have to handle orthogonal reliability issues
such as external database, time virtualization, and other examples
from our previous post?

> USERSPACE GLUE  possible                        possible
>

> SECURITY root and non-root modes root and non-root modes
> native support for LSM
>
> MAINTENANCE changes mainly for features changes mainly for features;
> create new ABI for features

> iAnd by all means, I intend to cooperate with Gene to see how to
> make the other part of DMTCP, namely the userspace "glue", work on
> top of linux-cr to have the benefits of all worlds !

This is true, and we strongly welcome the cooperation. We don't know how this experiment will turn out, but the only way to find out is to sincerely try it. Whether we succeed or fail, we will learn something either way!

- Gene and Kapil

_____

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