

In this post, Kapil and I will provide our own summary of how we see the issues for discussion so far. In the next post, we'll reply specifically to comment on Oren's table of comparison between linux-cr and userspace.

In general, we'd like to add that the conversation with Oren was very useful for us, and I think Oren will also agree that we were able to converge on the purely technical questions.

Concerning opinions, we want to be cautious on opinions, since we're still learning the context of this ongoing discussion on LKML. There is probably still some context that we're missing.

Below, we'll summarize the four major questions that we've understood from this discussion so far. But before doing so, I want to point out that a single process or process tree will always have many possible interactions with the rest of the world. Within our own group, we have an internal slogan:

"You can't checkpoint the world."

A virtual machine can have a relatively closed world, which makes it more robust, but checkpointing will always have some fragile parts.

We give four examples below:

- a. time virtualization
- b. external database
- c. NSCD daemon
- d. screen and other full-screen text programs

These are not the only examples of difficult interactions with the rest of the world.

Anyway, in my opinion, the conversation with Oren seemed to converge into two larger cases:

1. In a pure userland C/R like DMTCP, how many corner cases are not handled, or could not be handled, in a pure userland approach?

Also, how important are those corner cases? Do some have important use cases that rise above just a corner case?

[ inotify is one of those examples. For DMTCP to support this, it would have to put wrappers around inotify\_add\_watch, inotify\_rm\_watch, read, etc., and maybe even tracking inodes in case the file had been renamed after the inotify\_add\_watch. Something could be made to work for the common cases, but it would still be a hack --- to be done only if a use case demands it. ]

2. In a Linux C/R approach, it's already recognized that one needs a userland component (for example, for convenience of recreating the process tree on restart). How many other cases are there that require a userland component?

[ One example here is the shared memory segment of NSCD, which has to be re-initialized on restart. Another example is a screen process that talks to an ANSI terminal emulator (e.g. gnome-terminal), which talks to an X server or VNC server. Below, we discuss these examples in more detail. ]

One can add a third and fourth question here:

3. [Originally posed by Oren] Given Linux C/R, how much work would it be to add the higher layers of DMTCP on top of Linux C/R?  
[ This is a non-trivial question. As just one example, DMTCP handles sockets uniformly, regardless of whether they are intra-host or inter-host. Linux C/R handles certain types of intra-host sockets. So, merging the two would require some thought. ]
4. [Originally posed by Tejun, e.g. Fri Nov 19 2010 - 09:04:42 EST] Given that DMTCP checkpoints many common applications, how much work would it be to add a small number of restricted kernel interfaces to enable one to remove some of the hacks in DMTCP, and to cover the more important corner cases that DMTCP might be missing?

I'd also like to add some points of my own here. First, there are certain cases where I believe that a checkpoint-restart system (in-kernel or userland or hybrid) can never be completely transparent. It's because you can't completely cut the connection with the rest of the world. In these examples, I'm thinking primarily of the Linux C/R mode used to checkpoint a tree of processes.

To the extent that Linux C/R is used with containers, it seems to me to be closer to lightweight virtualization. From there, I've seen that the conversation goes to comparing lightweight virtualization versus traditional virtual machines, but that discussion goes beyond my own personal expertise.

Here are some examples that I believe that every checkpointing system would suffer from the syndrome of trying to "checkpoint the world".

1. Time virtualization --- Right now, neither system does time virtualization. Both systems could do it. But what is the right policy?

For example, one process may set a deadline for a task an hour in the future, and then periodically poll the kernel for the current time to see if one hour has passed. This use case seems to require time virtualization.

A second process wants to know the current day and time, because a certain web service updates its information at midnight each day. This use case seems to argue that time virtualization is bad.

2. External database file on another host --- It's not possible to

checkpoint the remote database file. In our work with the Condor developers, they asked us to add a "Condor mode", which says that if there are any external socket connections, then delay the checkpoint until the external socket connections are closed. In a different joint project with CERN (Geneva), we considered a checkpointing application in which an application saves much of the database, and then on restart, discovers how much of its data is stale, and re-loads only the stale portion.

3. NSCD (Network Services Caching Daemon) --- Glibc arranges for certain information to be cached in the NSCD. The information is in a memory segment shared between the NSCD and the application. Upon restart, the application doesn't know that the memory segment is no longer shared with the NSCD, or that the information is stale. The DMTCP "hack" is to zero out this memory page on restart. Then glibc recognizes that it needs to create a new shared memory segment.

3. screen --- The screen application sets the scrolling region of its ANSI terminal emulator, in order to create a status line at the bottom, while scrolling the remaining lines of the terminal. Upon restart, screen assumes that the scrolling region has already been set up, and doesn't have to be re-initialized. So, on restart, DMTCP uses SIGWINCH to fool screen (or any full-screen text-based application) into believing that its window size has been changed. So, screen (or vim, or emacs) then re-initializes the state of its ANSI terminal, including scrolling regions and so on.

So, a userland component is helpful in doing the kind of hacks above. I recognize that the Linux C/R team agrees that some userland component can be useful. I just want to show why some userland hacks will always be needed. Let's consider a pure in-kernel approach to checkpointing 'screen' (or almost any full-screen application that uses a status bar at the bottom). Screen sets the scrolling region of an ANSI terminal emulator, which might be a gnome-terminal. So, a pure in-kernel approach needs to also checkpoint the gnome-terminal. But the gnome-terminal needs to talk to an X server. So, now one also needs to start up inside a VNC server to emulate the X server. So, either one adds a "hack" in userland to force screen to re-initialize its ANSI terminal emulator, or else one is forced to include an entire VNC server just to checkpoint a screen process. ]

Finally, this excerpt below from Tejun's post sums up our views too. We don't have the kernel expertise of the people on this list, but we've had to do a little bit of reading the kernel code where the documentation was sparse and in teaching O/S. We would certainly be very happy to work closely with the kernel developers, if there was interest in extending DMTCP to directly use more kernel support.

- Gene and Kapil

Tejun Heo wrote Fri Nov 19 2010 - 09:04:42 EST

> What's so wrong with Gene's work? Sure, it has some hacky aspects but  
> let's fix those up. To me, it sure looks like much saner and  
> manageable approach than in-kernel CR. We can add nested ptrace,  
> CLONE\_SET\_PID (or whatever) in pidns, integrate it with various ns  
> supports, add an ability to adjust brk, export inotify state via  
> fdinfo and so on.  
>  
> The thing is already working, the codebase of core part is fairly  
> small and condor is contemplating integrating it, so at least some  
> people in HPC segment think it's already viable. Maybe the HPC  
> cluster I'm currently sitting near is special case but people here  
> really don't run very fancy stuff. In most cases, they're fairly  
> simple (from system POV) C programs reading/writing data and burning a  
> \_LOT\_ of CPU cycles inbetween and admins here seem to think dmtcp  
> integrated with condor would work well enough for them.  
>  
> Sure, in-kernel CR has better or more reliable coverage now but by how  
> much? The basic things are already there in userland.

---

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

---

---

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 conclusions.

> 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 handles 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/containers>

---



Subject: Re: [Ksummit-2010-discuss] checkpoint-restart: naked patch  
Posted by [Grant Likely](#) on Sun, 21 Nov 2010 22:41:35 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Sun, Nov 21, 2010 at 03:18:53AM -0500, Gene Cooperman wrote:

> In this post, Kapil and I will provide our own summary of how we  
> see the issues for discussion so far. In the next post, we'll reply  
> specifically to comment on Oren's table of comparison between  
> linux-cr and userspace.  
>  
> In general, we'd like to add that the conversation with Oren was very  
> useful for us, and I think Oren will also agree that we were able to  
> converge on the purely technical questions.

Hi Gene,

Thanks for the good summary, it helps. Some random comments below...

>  
> Concerning opinions, we want to be cautious on opinions, since we're  
> still learning the context of this ongoing discussion on LKML. There is  
> probably still some context that we're missing.  
>  
> Below, we'll summarize the four major questions that we've understood from  
> this discussion so far. But before doing so, I want to point out that a single  
> process or process tree will always have many possible interactions with  
> the rest of the world. Within our own group, we have an internal slogan:  
> "You can't checkpoint the world."  
> A virtual machine can have a relatively closed world, which makes it more  
> robust, but checkpointing will always have some fragile parts.  
> We give four examples below:  
> a. time virtualization  
> b. external database  
> c. NSCD daemon  
> d. screen and other full-screen text programs  
> These are not the only examples of difficult interactions with the  
> rest of the world.  
>  
> Anyway, in my opinion, the conversation with Oren seemed to converge  
> into two larger cases:  
> 1. In a pure userland C/R like DMTCP, how many corner cases are not handled,  
> or could not be handled, in a pure userland approach?  
> Also, how important are those corner cases? Do some  
> have important use cases that rise above just a corner case?  
> [ inotify is one of those examples. For DMTCP to support this,  
> it would have to put wrappers around inotify\_add\_watch,  
> inotify\_rm\_watch, read, etc., and maybe even tracking inodes in case  
> the file had been renamed after the inotify\_add\_watch. Something  
> could be made to work for the common cases, but it would

> still be a hack --- to be done only if a use case demands it. ]

> 2. In a Linux C/R approach, it's already recognized that one needs

> a userland component (for example, for convenience of recreating

> the process tree on restart). How many other cases are there

> that require a userland component?

> [ One example here is the shared memory segment of NSCD, which

> has to be re-initialized on restart. Another example is

> a screen process that talks to an ANSI terminal emulator

> (e.g. gnome-terminal), which talks to an X server or VNC server.

> Below, we discuss these examples in more detail. ]

>

> One can add a third and fourth question here:

>

> 3. [Originally posed by Oren] Given Linux C/R, how much work would

> it be to add the higher layers of DMTCP on top of Linux C/R?

> [ This is a non-trivial question. As just one example, DMTCP

> handles sockets uniformly, regardless of whether they

> are intra-host or inter-host. Linux C/R handles certain

> types of intra-host sockets. So, merging the two would

> require some thought. ]

> 4. [Originally posed by Tejun, e.g. Fri Nov 19 2010 - 09:04:42 EST]

> Given that DMTCP checkpoints many common applications, how much work

> would it be to add a small number of restricted kernel interfaces

> to enable one to remove some of the hacks in DMTCP, and to cover

> the more important corner cases that DMTCP might be missing?

>

>

> I'd also like to add some points of my own here. First, there are certain

> cases where I believe that a checkpoint-restart system (in-kernel

> or userland or hybrid) can never be completely transparent. It's because you

> can't completely cut the connection with the rest of the world. In these

> examples, I'm thinking primarily of the Linux C/R mode used to checkpoint

> a tree of processes.

> To the extent that Linux C/R is used with containers, it seems

> to me to be closer to lightweight virtualization. From there, I've

> seen that the conversation goes to comparing lightweight virtualization

> versus traditional virtual machines, but that discussion goes beyond my

> own personal expertise.

At the risk of restating already applied arguments, and as a c/r outsider, this touches on the real crux of the issue for me. What is the complete set of boundaries between a c/r group of processes and the outside world? Is it bounded and is it understandable by mere kernel engineers? Does it change the assumptions about what a Linux process *is*, and how to handle it? How much? The broad strokes seem to be straight forward, but as already pointed out, the devil is in the details.



- > Here are some examples that I believe that every checkpointing system
- > would suffer from the syndrome of trying to "checkpoint the world".
- >
- > 1. Time virtualization --- Right now, neither system does time virtualization.
- > Both systems could do it. But what is the right policy?
- > For example, one process may set a deadline for a task an hour
- > in the future, and then periodically poll the kernel for the current time
- > to see if one hour has passed. This use case seems to require time
- > virtualization.
- > A second process wants to know the current day and time, because a certain
- > web service updates its information at midnight each day. This use case seems
- > to seem to argue that time virtualization is bad.

Temporal issues need to be (are being?) addressed regardless. In certain respects, I'm sure c/r can be seen as a \*really long\* scheduler latency, and would have the same effect as a system going into suspend, or a vm-level checkpoint. I would think the same behaviour would be desirable in all cases, include c/r.

- > 2. External database file on another host --- It's not possible to
- > checkpoint the remote database file. In our work with the Condor developers,
- > they asked us to add a "Condor mode", which says that if there are any
- > external socket connections, then delay the checkpoint until the external
- > socket connections are closed. In a different joint project with CERN (Geneva),
- > we considered a checkpointing application in which an application
- > saves much of the database, and then on restart, discovers how much
- > of its data is stale, and re-loads only the stale portion.
- >
- > 3. NSCD (Network Services Caching Daemon) --- Glibc arranges for
- > certain information to be cached in the NSCD. The information is
- > in a memory segment shared between the NSCD and the application.
- > Upon restart, the application doesn't know that the memory segment
- > is no longer shared with the NSCD, or that the information is stale.
- > The DMTCP "hack" is to zero out this memory page on restart. Then glibc
- > recognizes that it needs to create a new shared memory segment.

Right here is exactly the example of a boundary that needs explicit rules. When a pair of processes have a shared region, and only one of them is checkpointed, then what is the behaviour on restore? In this specific example, a context-specific hack is used to achieve the desired result, but that doesn't work (as I believe you agree) in the general case. What behaviour will in-kernel support need to enforce?

- > 3. screen --- The screen application sets the scrolling region of
- > its ANSI terminal emulator, in order to create a status line
- > at the bottom, while scrolling the remaining lines of the terminal.
- > Upon restart, screen assumes that the scrolling region
- > has already been set up, and doesn't have to be re-initialized.

> So, on restart, DMTCP uses SIGWINCH to fool screen (or any  
> full-screen text-based application) into believing that its  
> window size has been changed. So, screen (or vim, or emacs)  
> then re-initializes the state of its ANSI terminal, including  
> scrolling regions and so on.

> So, a userland component is helpful in doing the kind of hacks above.  
> I recognize that the Linux C/R team agrees that some userland component  
> can be useful. I just want to show why some userland hacks will always be  
> needed. Let's consider a pure in-kernel approach to checkpointing 'screen'  
> (or almost any full-screen application that uses a status bar at the bottom).  
> Screen sets the scrolling region of an ANSI terminal emulator,  
> which might be a gnome-terminal. So, a pure in-kernel approach  
> needs to also checkpoint the gnome-terminal. But the gnome-terminal  
> needs to talk to an X server. So, now one also needs to start  
> up inside a VNC server to emulate the X server. So, either  
> one adds a "hack" in userland to force screen to re-initialize  
> its ANSI terminal emulator, or else one is forced to include  
> an entire VNC server just to checkpoint a screen process. ]

>

> Finally, this excerpt below from Tejun's post sums up our views too. We don't  
> have the kernel expertise of the people on this list, but we've had  
> to do a little bit of reading the kernel code where the documentation  
> was sparse and in teaching O/S. We would certainly be very happy to work  
> closely with the kernel developers, if there was interest in extending  
> DMTCP to directly use more kernel support.

>

> - Gene and Kapil

>

> Tejun Heo wrote Fri Nov 19 2010 - 09:04:42 EST

> > What's so wrong with Gene's work? Sure, it has some hacky aspects but  
> > let's fix those up. To me, it sure looks like much saner and  
> > manageable approach than in-kernel CR. We can add nested ptrace,  
> > CLONE\_SET\_PID (or whatever) in pidns, integrate it with various ns  
> > supports, add an ability to adjust brk, export inotify state via  
> > fdinfo and so on.

> >

> > The thing is already working, the codebase of core part is fairly  
> > small and condor is contemplating integrating it, so at least some  
> > people in HPC segment think it's already viable. Maybe the HPC  
> > cluster I'm currently sitting near is special case but people here  
> > really don't run very fancy stuff. In most cases, they're fairly  
> > simple (from system POV) C programs reading/writing data and burning a  
> > \_LOT\_ of CPU cycles inbetween and admins here seem to think dmtcp  
> > integrated with condor would work well enough for them.

> >

> > Sure, in-kernel CR has better or more reliable coverage now but by how  
> > much? The basic things are already there in userland.

> --

- > To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
- > the body of a message to [majordomo@vger.kernel.org](mailto:majordomo@vger.kernel.org)
- > More majordomo info at <http://vger.kernel.org/majordomo-info.html>
- > Please read the FAQ at <http://www.tux.org/lkml/>

---

Containers mailing list

[Containers@lists.linux-foundation.org](mailto:Containers@lists.linux-foundation.org)

<https://lists.linux-foundation.org/mailman/listinfo/containers>

---