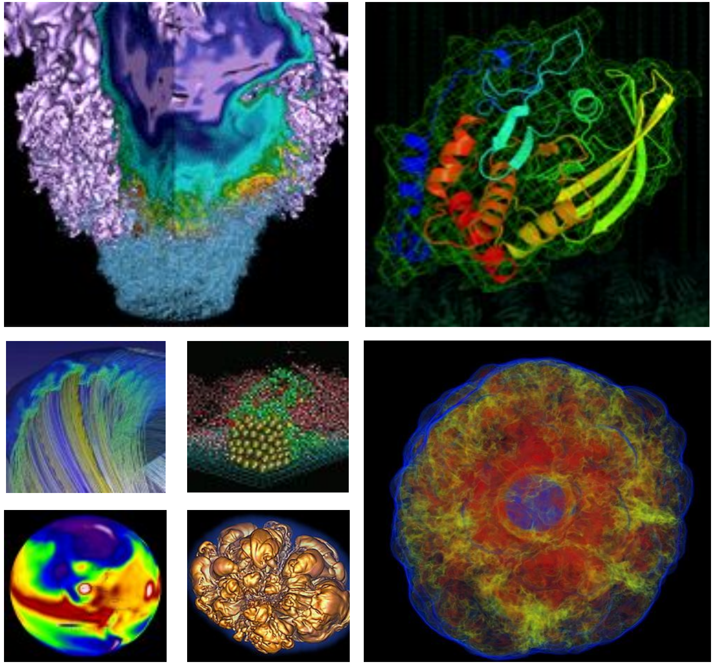
Checkpointing and Restarting Jobs with DMTCP Inside the Container



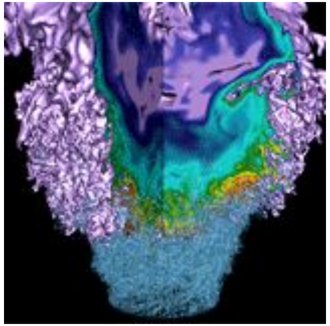
NERSC Data Day
Feb 21-22 2024

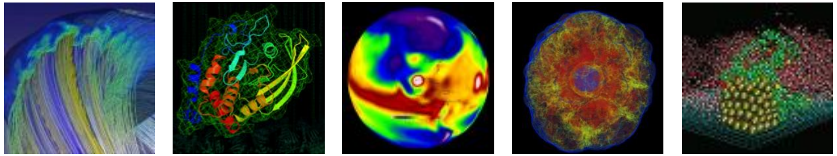
Madan Timalsina
NERSC/NESAP Postdoc
Data & AI Services

Outline

- Introduction
- DMTCP (Distributed MultiThreaded CheckPointing) Overview
- Checkpointing and Restarting Jobs using DMTCP within Containers on Perlmutter
- Conclusion

Introduction





Checkpointing and Restarting (C/R)

- ***Checkpointing*** involves preserving the current state of a running process (jobs) by creating a checkpoint image file.
  - This includes capturing the memory, executing instructions, I/O status, and related data of the running process into a file.
- ***Restarting*** the process is possible using the checkpoint file.
  - This enables the process to resume its execution from where it was saved (rather than from the beginning), either on the same or a different computer, seamlessly continuing its operation.

It's a crucial capability in High-Performance Computing (HPC) due to complex and time-consuming computations. It can reduce startup times in applications and facilitates batch scheduler optimizations, including preemption.

C/R: Benefits

NERSC Perspective

- **Enhanced Job Prioritization:** Potential preempting of less critical jobs for more urgent or time-sensitive tasks.
- **Optimized Node Utilization:** Efficient backfilling, maximizing node usage, especially for large reservations.
- **Uninterrupted Operations:** Run checkpointing jobs until system maintenance, ensuring minimal disruption.
- **Enhanced Reliability:** Potentially checkpointing all jobs before unexpected power outages for system stability and job recovery.

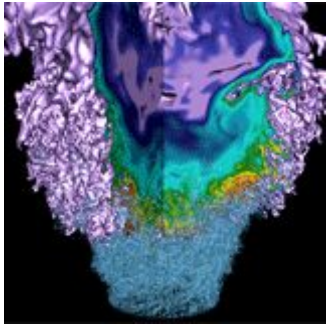
User Perspective

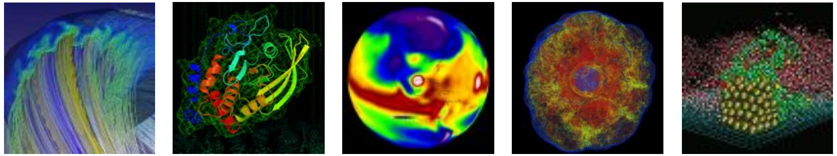
- **Extended Runtime:** Allow jobs to exceed walltime limits by resuming from checkpoints.
- **Increased Throughput:** Leveraging gaps in the Slurm schedule to optimize job processing.
- **Extended Interactivity:** Save and resume interactive sessions seamlessly (if it's time to go home to dinner, then checkpoint and restart the next day!*)*
- **Efficient Debugging**: Pause, identify errors, and restart jobs from specific checkpoints for iterative debugging.

Challenges in C/R

- ***Complexity for User Transparency:*** Requires extensive effort to create a seamless experience for users during checkpointing and restarting processes.
- ***MPI Support Challenges:*** Particularly intricate due to the combination of various MPI implementations (e.g., MPICH, OpenMPI) and networks (e.g., ethernet, Cray Aries), resulting in the need for multiple versions (MxN problem).
- DMTCP serves as a solution for overcoming these challenges.
- For more details, refer to the NERSC documentation

DMTCP: Distributed MultiThreaded CheckPointing





NERSC documentation, DMTCP website, DMTCP github

The DMTCP project is partially supported by grants from Intel Corporation, and from the National Science Foundation under grants OCI-0960978, ACI-1440788, and OAC-1740218.

DMTCP: Simplifying Checkpoint-Restart

An open-source tool offering seamless checkpoint and restart functionalities for distributed applications across clusters, grids, cloud environments etc.

Preserves Application State Seamlessly

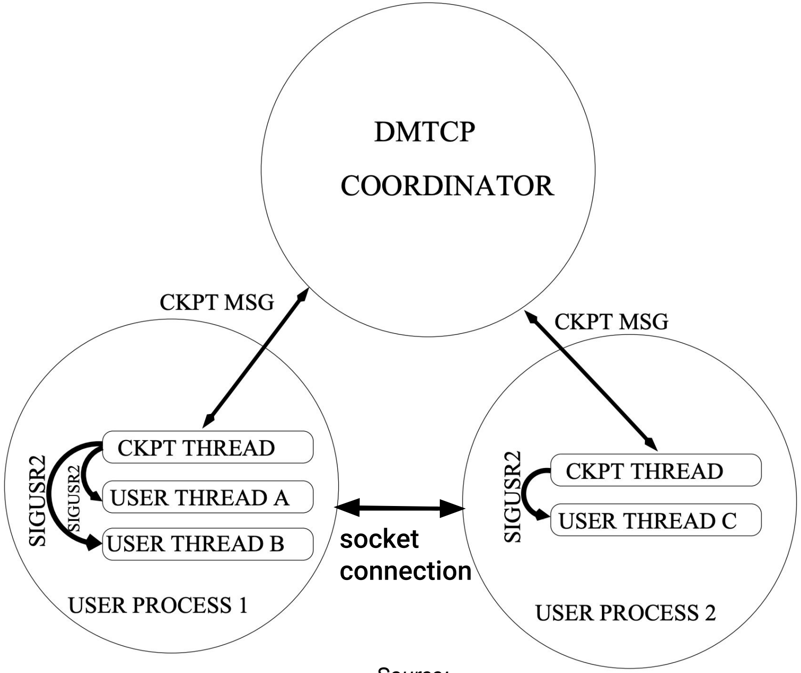
- **No Code or Kernel Modifications:** Stores complex threaded or distributed applications without altering their code or the Linux kernel.
- **Accessible to Users:** Doesn't require special system privileges, allowing operation without root access.

User-Friendly Checkpointing

- **Seamless User-Space Operation:** Performs checkpoints without changing user code or system settings.
- **Versatile Application Support:** Works with diverse applications like MPI, OpenMP, MATLAB, Python, C/C++/Fortran, shell scripts, and resource managers (e.g., Slurm).

How does DMTCP Work?

DMTCP Architecture: Coordinated Checkpointing



Source: Journal of Physics: Conference Series, 523(1):012015, june 2014

DMTCP Coordinator to Computation Ratio: One DMTCP coordinator manages one checkpointable DMTCP computation.

Multiple Checkpointable Computations: Multiple coordinators can handle separate computations, each independently checkpointable.

Checkpoint Thread vs. User Thread: Only one of the DMTCP checkpoint thread or user thread can be active at any given time, not both concurrently.

Fault Tolerance without Single Point of Failure: No single point of failure if checkpoint image files are backed up. Even if the coordinator fails, the system can restart from the last checkpoint.

Preservation of Runtime Libraries: Runtime libraries are saved as part of the memory image. Applications continue using the same library API.

Inclusion of Linux Environment Variables: Linux environment variables are part of the memory image. Special DMTCP plugin needed to modify saved environment variables during checkpoint.

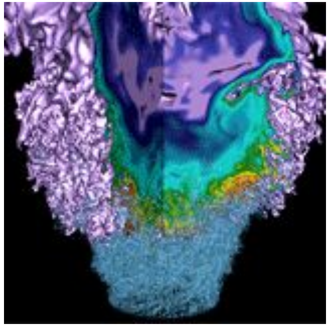
User-Space Functionality: Entire process operates in user-space; no need for administrative privileges for its functioning.

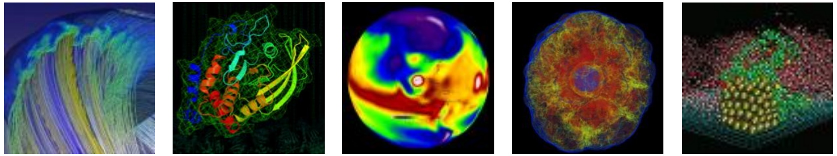
RESTART: same as ckpt, but in opposite order

Checkpoint/Restart (C/R) Integration Using DMTCP

- Conducted different tests across multiple versions of Geant4 (10.5, 10.7, and 11.0) for a variety of simulations.
- Geant4 is a crucial tool for High Energy Physics (HEP) research, has been thoroughly tested and has passed the assessments.
- Performed tests using Shifter and Podman-HPC container images.
- Planning to extend our research into additional fields such as material science, with ongoing tests using CP2K.

Checkpoint/Restart (C/R) Jobs inside Container using DMTCP: Perlmutter





SHIFTER

NERSC

podman

U.S. DEPARTMENT OF ENERGY | Office of Science

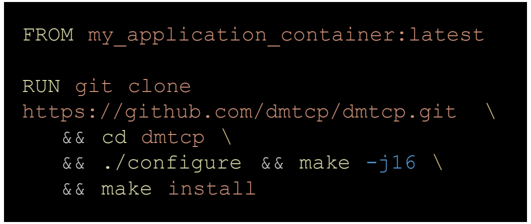
BERKELEY LAB
Lawrence Berkeley National Laboratory

Requirements:

- DMTCP cannot be checkpointed from outside the containers. It must be included within the container when it is build.

- The simulation package can be built in many ways:
  - During the container's build process.
  - After the container has been built, by linking the source code from elsewhere.
  - Extend the functionality by building on top of an existing container, enabling quick experimentation with minimal modifications.

  All methods have been tested and verified.



```
FROM my_application_container:latest

RUN git clone
https://github.com/dmtcp/dmtcp.git  \
  && cd dmtcp \
  && ./configure && make -j16 \
  && make install
```

- In the context of Geant4, various versions can be directly sourced from the CernVM File System (CVMFS), facilitating easy access to multiple versions for testing and deployment.

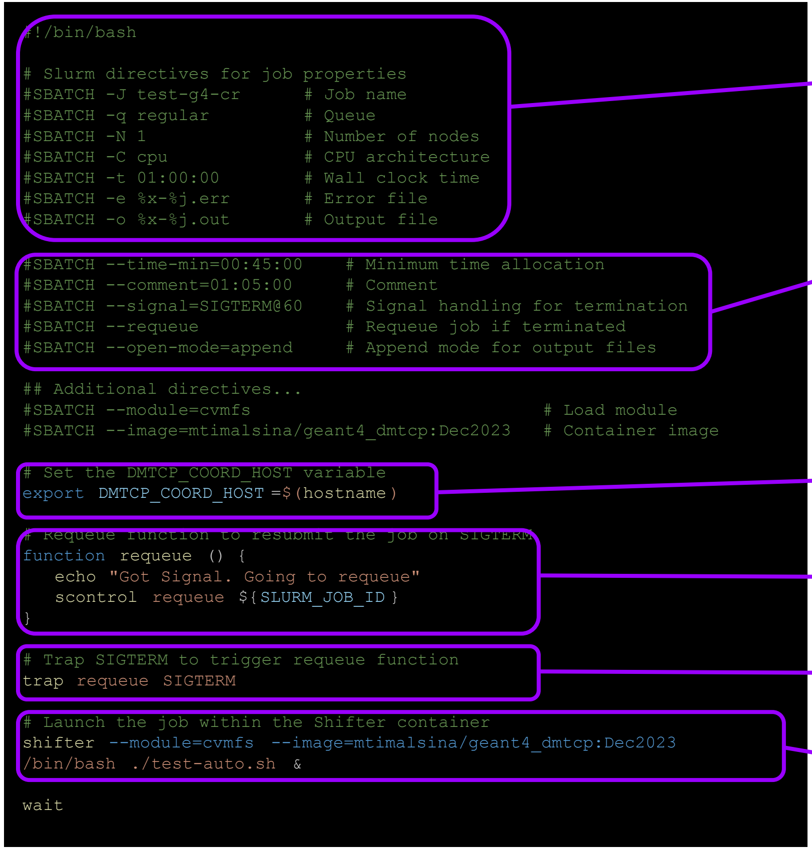
How Does Automatic Resubmission of Jobs Work?

- Users submit their job scripts, incorporating DMTCP within containers, along with necessary software packages like Geant4, CP2K.
- A tailored script is used to manage checkpoint-restart tasks, which isn't directly feasible within the container environment.
- The script initiates checkpointing via *restart_job* function including a *start_coordinator* to initiate jobs and executes using *dmtcp_launch*, ensuring efficient job lifecycle management.
- Upon receiving termination signals *(SIGTERM)*, the setup facilitates checkpointing, ensuring continuous job execution and effective resource utilization.
- This method ensures efficient handling of Checkpoint/Restart processes, aligning with the specific needs of HPC environments, leading to the successful completion of jobs.

C/R Jobs with DMTCP within Container: Perlmutter

To run:

```
sbatch run.sh
```



```
#!/bin/bash

# Slurm directives for job properties
#SBATCH -J test-g4-cr      # Job name
#SBATCH -q regular         # Queue
#SBATCH -N 1               # Number of nodes
#SBATCH -C cpu             # CPU architecture
#SBATCH -t 01:00:00        # Wall clock time
#SBATCH -e %x-%j.err       # Error file
#SBATCH -o %x-%j.out       # Output file

#SBATCH --time-min=00:45:00    # Minimum time allocation
#SBATCH --comment=01:05:00     # Comment
#SBATCH --signal=SIGTERM@60    # Signal handling for termination
#SBATCH --requeue              # Requeue job if terminated
#SBATCH --open-mode=append     # Append mode for output files

## Additional directives...
#SBATCH --module=cvmfs                             # Load module
#SBATCH --image=mtimalsina/geant4_dmtcp:Dec2023    # Container image

# Set the DMTCP_COORD_HOST variable
export DMTCP_COORD_HOST =$(hostname)

# Requeue function to resubmit the job on SIGTERM
function requeue () {
   echo "Got Signal. Going to requeue"
   scontrol requeue ${SLURM_JOB_ID}
}

# Trap SIGTERM to trigger requeue function
trap requeue SIGTERM

# Launch the job within the Shifter container
shifter --module=cvmfs --image=mtimalsina/geant4_dmtcp:Dec2023
/bin/bash ./test-auto.sh &

wait
```

Basic slurm directives

New for C/R jobs with DMTCP automatic resubmission

--comment sbatch flag is used to specify the desired walltime and to track the remaining walltime for the job after pre-termination

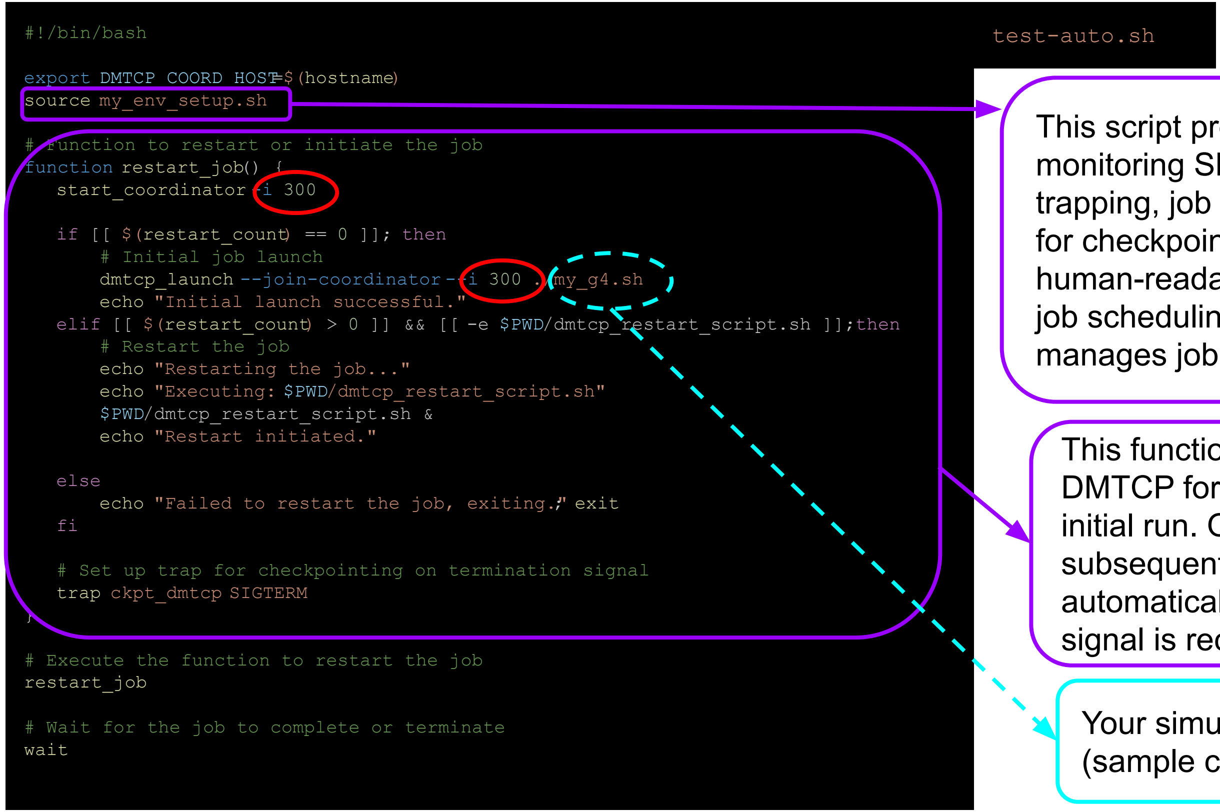
Export hostname to restart the job

Requeue function to resubmit the job

Trap signal (SIGTERM) to trigger requeue function

Launch the job within the Shifter container

C/R Jobs with DMTCP within Container: Perlmutter



test-auto.sh

```
#!/bin/bash

export DMTCP_COORD_HOST=$(hostname)
source my_env_setup.sh

# function to restart or initiate the job
function restart_job() {
   start_coordinator -i 300

   if [[ $(restart_count) == 0 ]]; then
       # Initial job launch
       dmtcp_launch --join-coordinator -i 300 ./my_g4.sh
       echo "Initial launch successful."
   elif [[ $(restart_count) > 0 ]] && [[ -e $PWD/dmtcp_restart_script.sh ]];then
       # Restart the job
       echo "Restarting the job..."
       echo "Executing: $PWD/dmtcp_restart_script.sh"
       $PWD/dmtcp_restart_script.sh &
       echo "Restart initiated."

   else
       echo "Failed to restart the job, exiting."; exit
   fi

   # Set up trap for checkpointing on termination signal
   trap ckpt_dmtcp SIGTERM
}

# Execute the function to restart the job
restart_job

# Wait for the job to complete or terminate
wait
```

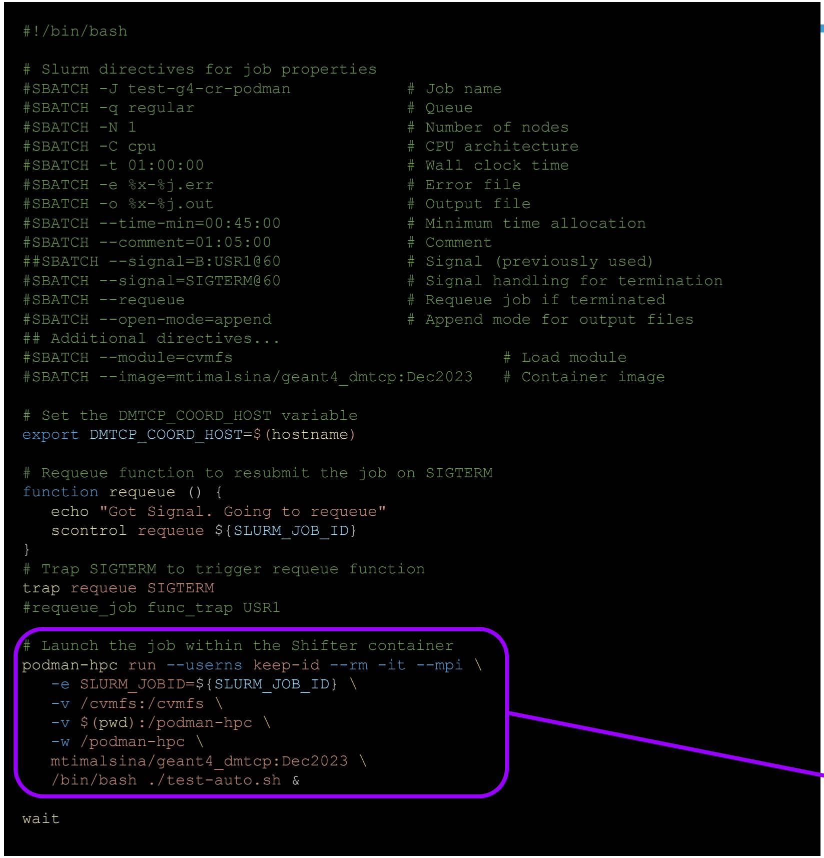
This script provides functions for managing and monitoring SLURM jobs, including time tracking, signal trapping, job requeuing, and integration with DMTCP for checkpoint/restart functionality. It converts time to human-readable format, calculates remaining time for job scheduling, updates job comments accordingly, and manages job requeuing based on the remaining time.

This function sets up and manages a job using DMTCP for checkpointing. It starts the job if it's the initial run. Or restarts it from a checkpoint if it's a subsequent run. Additionally, it configures a trap to automatically checkpoint the job when a termination signal is received.

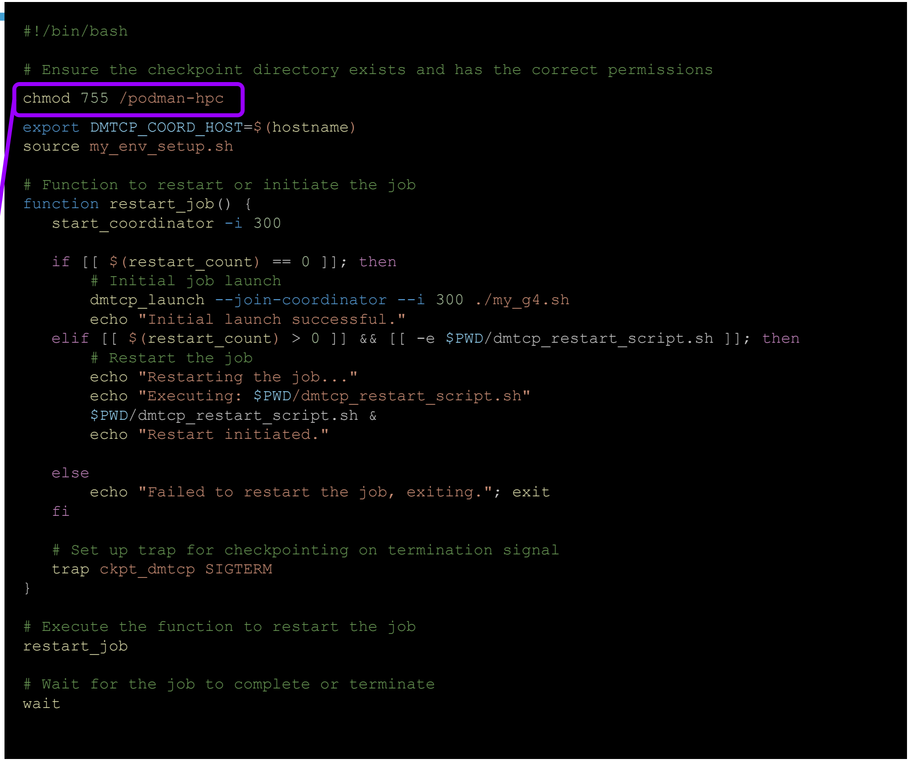
Your simulation code
(sample code is in backup slide)

Users can choose the checkpoint interval with the *-i* option.

C/R Jobs with DMTCP within Container: Perlmutter



```
#!/bin/bash

# Slurm directives for job properties
#SBATCH -J test-g4-cr-podman            # Job name
#SBATCH -q regular                      # Queue
#SBATCH -N 1                            # Number of nodes
#SBATCH -C cpu                          # CPU architecture
#SBATCH -t 01:00:00                     # Wall clock time
#SBATCH -e %x-%j.err                    # Error file
#SBATCH -o %x-%j.out                    # Output file
#SBATCH --time-min=00:45:00             # Minimum time allocation
#SBATCH --comment=01:05:00              # Comment
##SBATCH --signal=B:USR1@60             # Signal (previously used)
#SBATCH --signal=SIGTERM@60             # Signal handling for termination
#SBATCH --requeue                       # Requeue job if terminated
#SBATCH --open-mode=append              # Append mode for output files
## Additional directives...
#SBATCH --module=cvmfs                            # Load module
#SBATCH --image=mtimalsina/geant4_dmtcp:Dec2023   # Container image

# Set the DMTCP_COORD_HOST variable
export DMTCP_COORD_HOST=$(hostname)

# Requeue function to resubmit the job on SIGTERM
function requeue () {
   echo "Got Signal. Going to requeue"
   scontrol requeue ${SLURM_JOB_ID}
}
# Trap SIGTERM to trigger requeue function
trap requeue SIGTERM
#requeue_job func_trap USR1

# Launch the job within the Shifter container
podman-hpc run --userns keep-id --rm -it --mpi \
   -e SLURM_JOBID=${SLURM_JOB_ID} \
   -v /cvmfs:/cvmfs \
   -v $(pwd):/podman-hpc \
   -w /podman-hpc \
   mtimalsina/geant4_dmtcp:Dec2023 \
   /bin/bash ./test-auto.sh &

wait
```



```
#!/bin/bash

# Ensure the checkpoint directory exists and has the correct permissions
chmod 755 /podman-hpc
export DMTCP_COORD_HOST=$(hostname)
source my_env_setup.sh

# Function to restart or initiate the job
function restart_job() {
   start_coordinator -i 300

   if [[ $(restart_count) == 0 ]]; then
       # Initial job launch
       dmtcp_launch --join-coordinator --i 300 ./my_g4.sh
       echo "Initial launch successful."
   elif [[ $(restart_count) > 0 ]] && [[ -e $PWD/dmtcp_restart_script.sh ]]; then
       # Restart the job
       echo "Restarting the job..."
       echo "Executing: $PWD/dmtcp_restart_script.sh"
       $PWD/dmtcp_restart_script.sh &
       echo "Restart initiated."

   else
       echo "Failed to restart the job, exiting."; exit
   fi

   # Set up trap for checkpointing on termination signal
   trap ckpt_dmtcp SIGTERM
}

# Execute the function to restart the job
restart_job

# Wait for the job to complete or terminate
wait
```

Significant modifications have been implemented in the ***shifter*** image script to ensure compatibility with ***podman-hpc***

Conclusion:

- The study showcases the effectiveness of checkpoint-restart techniques using DMTCP in High-Performance Computing environments.
- Demonstrated utility across HPC platforms including container technologies like Shifter and Podman-HPC .
- This method is particularly valuable in complex, lengthy HPC computations, significantly reducing time and cost associated with process restarts.
- Implementation in diverse simulations including HEP, medical science, and material science (test ongoing), showcasing versatility.
- Highlights a critical advancement in efficient and reliable computational methodologies.
- Confirms the effectiveness of the technique and opens new opportunities in computational science.

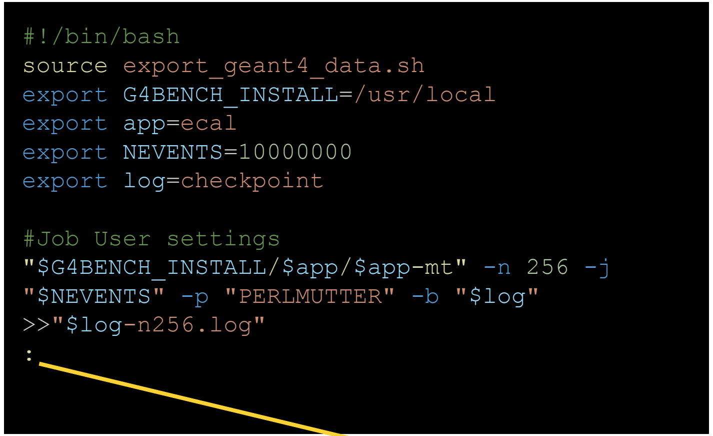
Thank You

Thanks: N. Tyler, L. Gerhardt, J. Blaschke, and W. Arndt

C/R Jobs with DMTCP within Container: Perlmutter

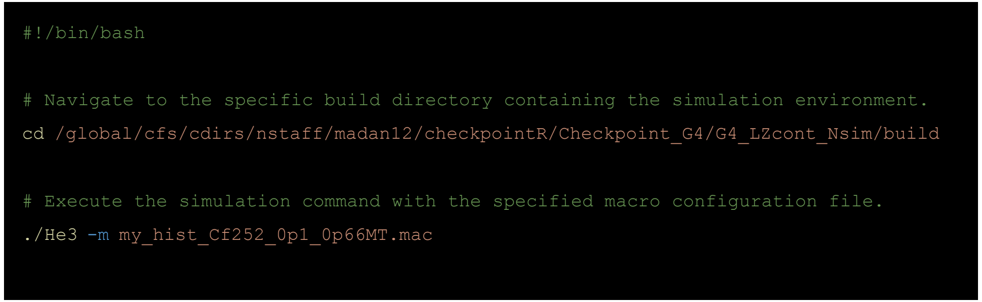
Here's my version of my_g4.sh, a simulation code

Case I: Compile the simulation code while building the container



```
#!/bin/bash
source export_geant4_data.sh
export G4BENCH_INSTALL=/usr/local
export app=ecal
export NEVENTS=10000000
export log=checkpoint

#Job User settings
"$G4BENCH_INSTALL/$app/$app-mt" -n 256 -j
"$NEVENTS" -p "PERLMUTTER" -b "$log"
>>"$log-n256.log"
:
```

Case II: Compile the simulation code inside the container after it's been built



```
#!/bin/bash

# Navigate to the specific build directory containing the simulation environment.
cd /global/cfs/cdirs/nstaff/madan12/checkpointR/Checkpoint_G4/G4_LZcont_Nsim/build

# Execute the simulation command with the specified macro configuration file.
./He3 -m my_hist_Cf252_0p1_0p66MT.mac
```

Added colon at end to counts as a noop command because dmtcp fails to recognize when the process has naturally ended

Some DMTCP Commands

dmtcp_coordinator -- coordinates checkpoints between multiple processes.

Example: `-i, --interval`: Time interval between automatic checkpoints (sec).

`--exit-on-last`: Auto-exits when the last client disconnects.

dmtcp_launch -- Start a process under DMTCP control.

Example: `-i, --interval`: Time interval between automatic checkpoints (sec).

`-j, --join-coordinator`: Join an existing coordinator, raise error if one doesn't already exist

dmtcp_restart -- Restart processes from a checkpoint image.

Example: `-h, --coord-host`: Specifies the hostname where `dmtcp_coordinator` is running

`-i, --interval`: Time interval between automatic checkpoints (sec).

dmtcp_command -- Send a command to the dmtcp_coordinator remotely.

Example: `-s --status`: Prints status message

`-k --kill`: Kills all nodes

`-q --quit`: Kills all nodes and quits

For more details, refer to the DMTCP website, NERSC documentation

C/R Jobs with DMTCP inside the Container

A tailored script

- Provides bash functions for managing Checkpoint/Restart (C/R) jobs

Starting the Coordinator

- Use the *start_coordinator* bash function, part of the tailored script.
- It executes the *dmtcp_launch* command with specific settings.
- Generates a *dmtcp_command.<jobid>* file in the run directory for job communication.

Coordinator Command Details

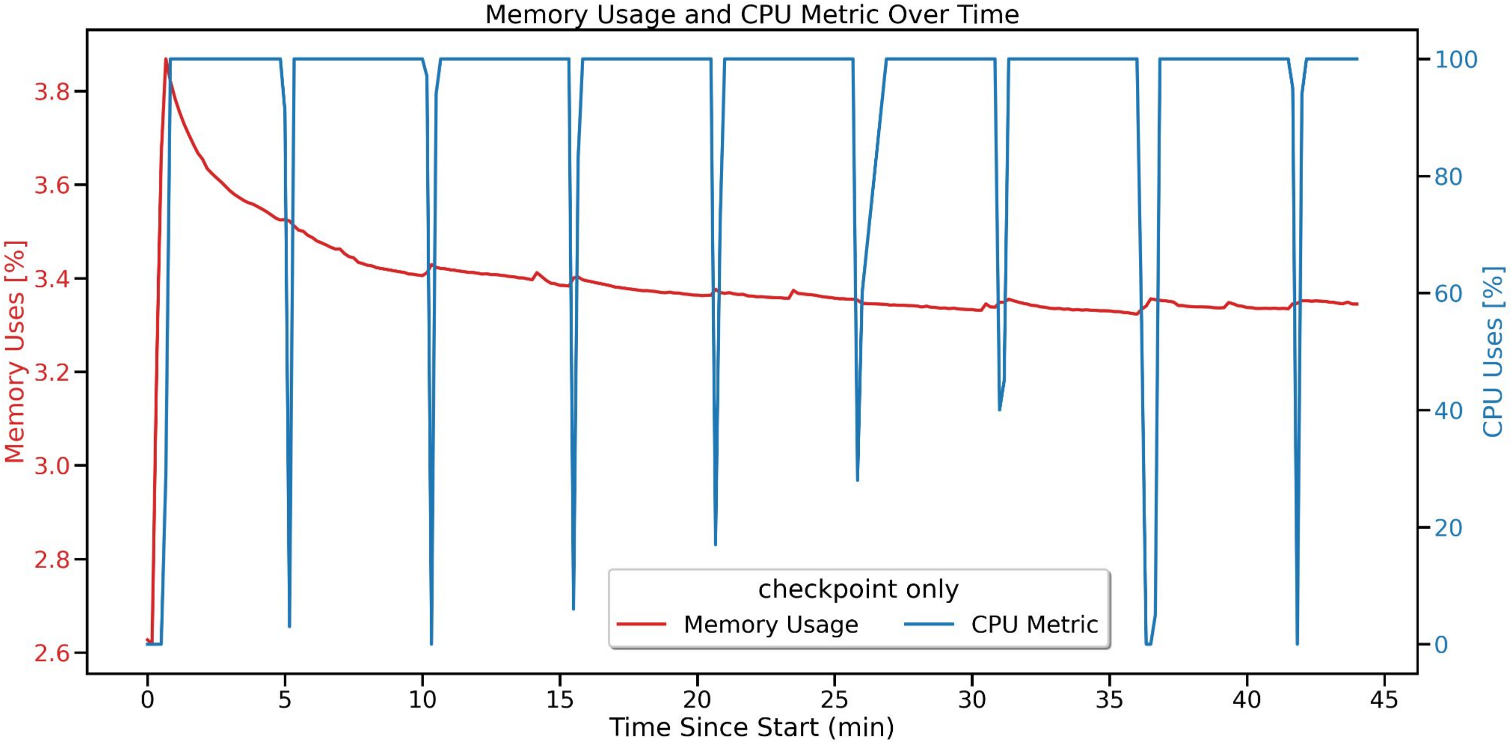
- Command: *dmtcp_coordinator --daemon --exit-on-last -p 0 --port-file $fname $@ 1>/dev/null 2>&1*
- Sets environment variables: (*export DMTCP_COORD_HOST=$h and export DMTCP_COORD_PORT=$p*)

Checkpoint Interval Selection

- Users can choose the checkpoint interval with the *-i* option.
- Options include periodic checkpoints or a single checkpoint before job termination.
- The checkpoint process overhead should be minimized, ideally less than the time to dump the node's full memory to disk.

C/R Jobs with DMTCP within Container: Perlmutter

Impact on total runtimes and memory footprint (Preliminary)



Thanks: Dhruva Kulkarni