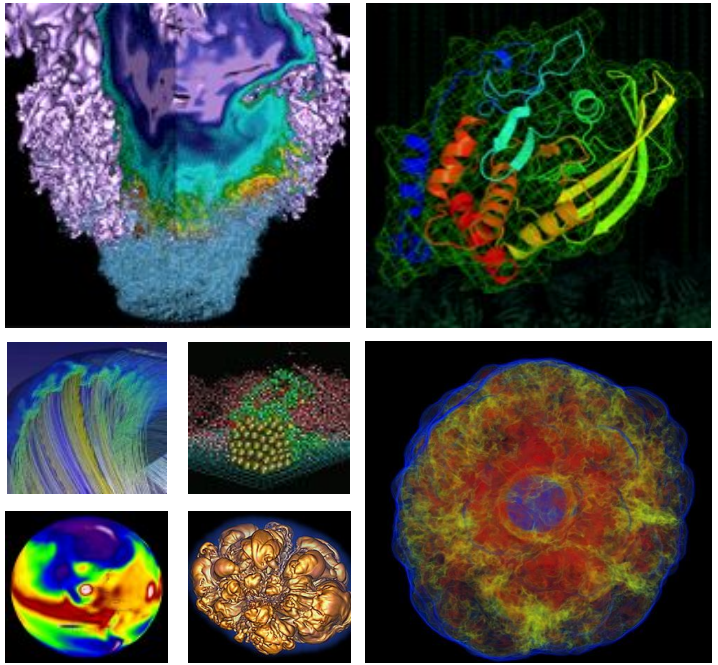
Containerized Checkpoint-Restart for HPC



SC24
Nov 17-22, 2024
Atlanta, USA

Presented at
CANOPIE-Workshop
Nov 17, 2024

Madan Timalsina

NERSC/NESAP Postdoc
Data & AI Services
LBNL

Co-presenter : N. Tyler (NERSC)

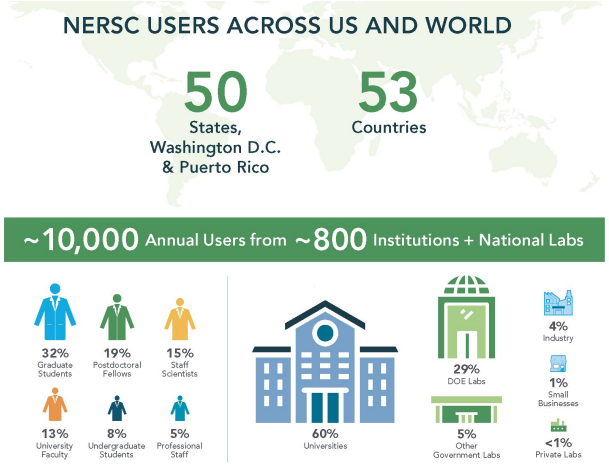
Thanks (NERSC): W. Arndt, J. P. Blaschke, L. Gerhardt

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

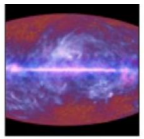
NERSC

- NERSC, the DOE Office of Science's premier facility for high-performance computing and data analysis
- Perlmutter, NERSC's flagship supercomputer, accelerates AI, data analysis, and simulations with its hybrid CPU-GPU architecture
- NERSC Science Acceleration Program (NESAP) facilitates collaboration with code teams, vendors, and developers to optimize scientific applications for cutting-edge computational architectures and next-generation supercomputing systems

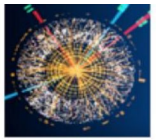


Acknowledged in ~ 5,800 refereed scientific publications & high profile journals since 2020

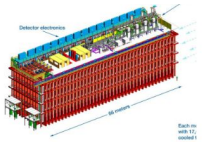

Palomar Transient Factory Supernova

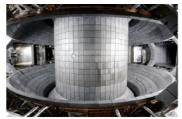

Planck Satellite Cosmic Microwave Background Radiation

Star Particle Physics


Atlas Large Hadron Collider


APS

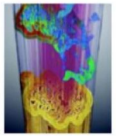

Dune


KStar


ARM


Katrin


Dayabay Neutrinos


ALS Light Source

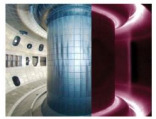

LCLS Light Source

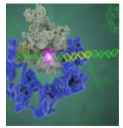

Joint Genome Institute Bioinformatics

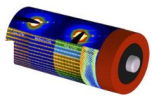

NSLS-II

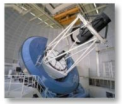

HSX


Majorana

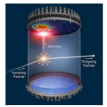

DIII-D


Cryo-EM


NCEM


DESI


LSST-DESC


LZ


IceCube


EXO

Containers

Containers are valuable to scientific computing users

- Encapsulation, isolation, portability, reproducibility, and even scalability

NERSC supports user container workloads via Shifter

- Developed at NERSC to address security concerns of docker
- Users can build their images with docker, then easily convert to shifter with a simple pull command

NERSC also supports podman-hpc

podman

- NERSC built wrapper for podman (open source tool)
- All the benefits of shifter, but using OCI (Open Container Initiative) standard runtime
- A rootless containers enhances security, users can build images at NERSC

Apptainer (formerly known as Singularity)

- Designed for high-performance computing, enhancing compatibility and security
- Facilitates seamless transitions between development, testing, and production without root access, maintaining adherence to the OCI standards
- Facilitates efficient management of containers through the CernVM File System

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

Checkpoint-Restart: Benefits

HPC/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

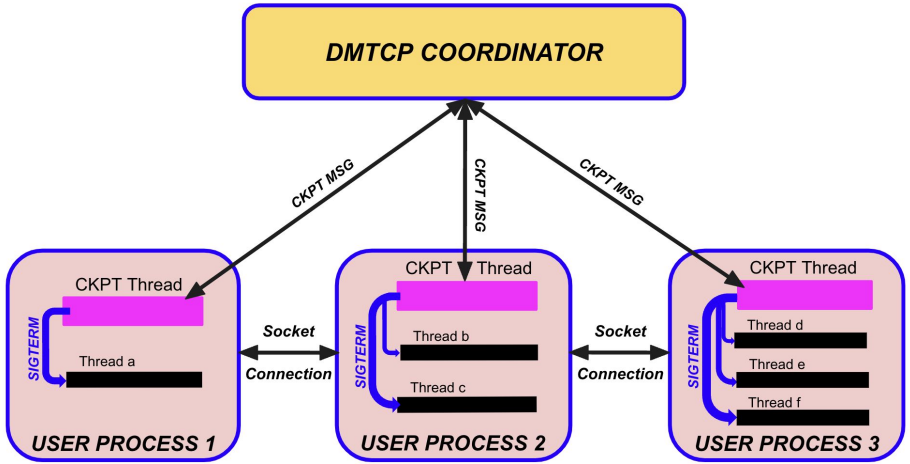
User Perspective

- **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

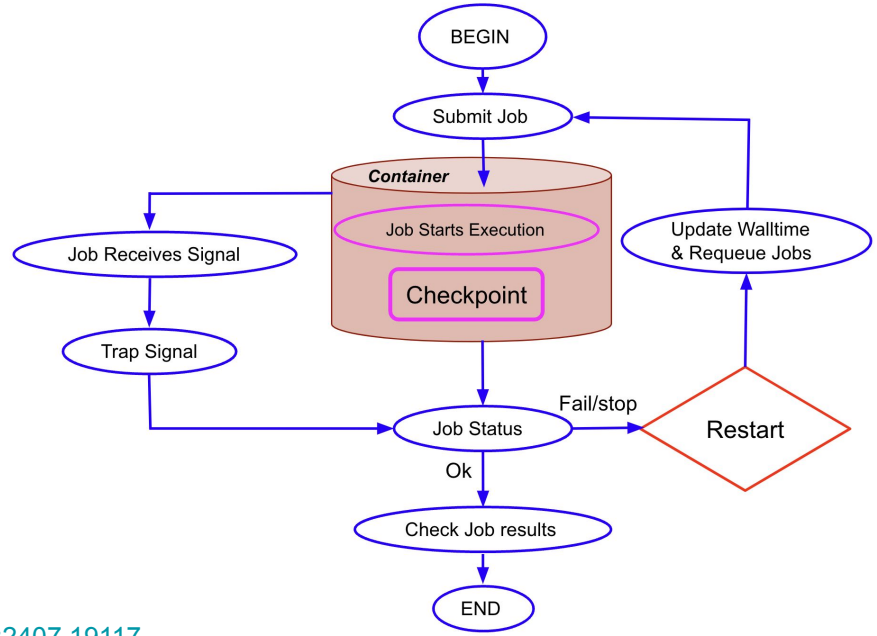
Using DMTCP within Containers for C/R

- DMTCP(Distributed MultiThreaded CheckPointing) is an open-source tool offering seamless checkpoint and restart functionalities for distributed applications across clusters, grids, cloud environments etc
  - No code or kernel modifications
  - Root access not required
  - OpenMP, Python, C/C++, Fortran, shell scripts, etc …

DMTCP Architecture:



- Users submit their job scripts, with the checkpoint interval (-i), incorporating DMTCP within containers, along with necessary software packages
- Helper scripts manage checkpoint-restart tasks, which isn't directly feasible within the container environment



arXiv:2407.19117

Requirements

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

- The software 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 DMTCP functionality by building on top of an existing container, enabling quick experimentation with minimal modifications

```
FROM my_application_container:latest
RUN git clone https://github.com/dmtcp/dmtcp.git \
    && cd dmtcp \
    && ./configure && make \
    && make install
```

All methods have been tested and verified

C/R Jobs with DMTCP within Container: Perlmutter

```
#!/bin/bash
# Slurm directives for job properties
...
```

Basic slurm directives

```
#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 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

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

Export hostname to restart the job

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

Requeue function to resubmit the job

```
# Trap SIGTERM to trigger requeue function
trap requeue SIGTERM
```

Trap signal (SIGTERM) to trigger requeue function

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

wait
```

Launch the job within the Shifter container

To run:

```
sbatch main.sh
```

C/R Jobs with DMTCP within Container: Perlmutter

podman-hpc

```
# Launch the job within the podman-hpc 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
```

Simple modifications to the main file are needed to ***launch*** the job within ***Podman-HPC*** and ***Apptainer***

Apptainer

```
# Launch the job within the Apptainer container
/cvmfs/oasis.opensciencegrid.org/mis/apptainer/
bin/apptainer exec -B /cvmfs:/cvmfs \
   $apptainer_image_path /bin/bash ./wrapper.sh &
wait
```

manage to run apptainer through the CernVM File System at NERSC Perlmutter

C/R Jobs with DMTCP within Container: Perlmutter

wrapper.sh

```
#!/bin/bash
export DMTCP_COORD_HOST=$(hostname)
source cr_env.sh

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

    if [[ $(restart_count) == 0 ]]; then
        # Initial job launch
        dmtcp_launch --join-coordinator -i 300 /example_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, time tracking, signal trapping, updating wall time, job requeuing, and integration with DMTCP for checkpoint/restart functionality.

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. It also configures a trap to automatically checkpoint the job when a termination signal is received

Your simulation code

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

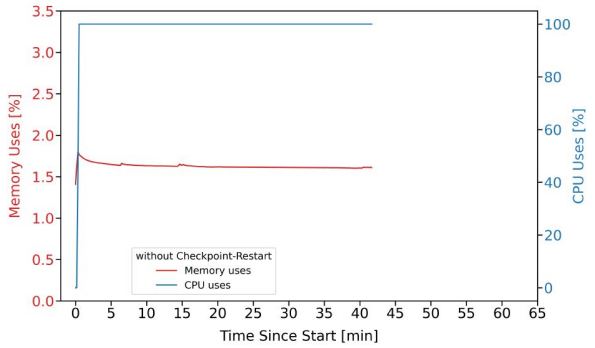
Results

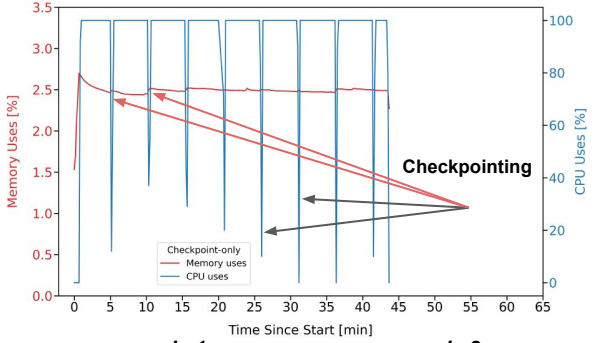
Impact of C/R on resource utilization

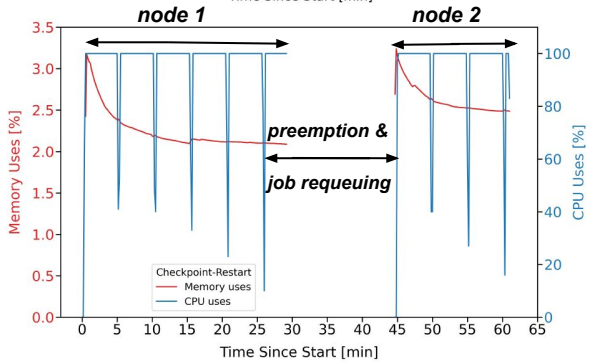
- ***Without C/R:*** The normal operational regime shows consistent CPU use and effective memory management
- ***Checkpoint-Only:*** Regular peaks in memory usage at checkpoints, with corresponding declines in CPU utilization
- ***Checkpoint-Restart:*** Spikes in memory use during checkpoints followed by corresponding declines in CPU utilization. A gaps in memory and CPU utilization due to preemption and job requeuing. We can see job has restarted in the different node afterward

C/R techniques exhibit a slight increase in computation time and memory usage (< 1%) because of DMTCP and associated file loading; however, this approach greatly reduces time and resource use by resuming the task from the last checkpoint state, enhancing efficiency

arXiv:2407.19117







Summary and Future Directions

- The study showcases the effectiveness of checkpoint-restart techniques using DMTCP within containerized High-Performance Computing (HPC) environments.
  - LZ Dark matter uses DMTCP C/R within container for production Neutron simulation
- Demonstrated utility across HPC platforms, including container technologies like Shifter, Podman-HPC, and Apptainer.
- This method is particularly valuable in complex, lengthy HPC computations, significantly reducing the time and cost associated with process restarts.
- DMTCP provides a robust checkpoint-restart solution for both single and multi-threaded applications.
- MANA (MPI-Agnostic Network-Agnostic) will be used as a plugin in DMTCP to improve checkpointing efficiency in MPI-based applications.
- Plans are underway to extend C/R techniques from high-energy physics (HEP) to other science domains
- Ongoing efforts focus on developing a GPU plugin for DMTCP, aiming to enable efficient checkpoint-restart in containerized GPU environments.

Thank You

Result

Check Science result with and without C/R

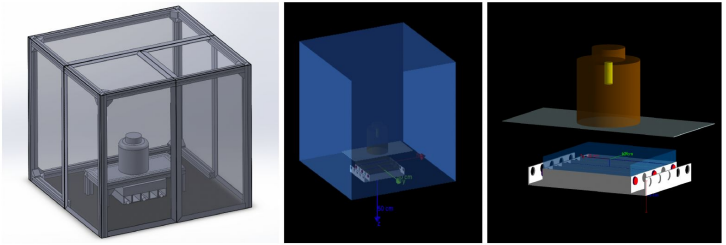
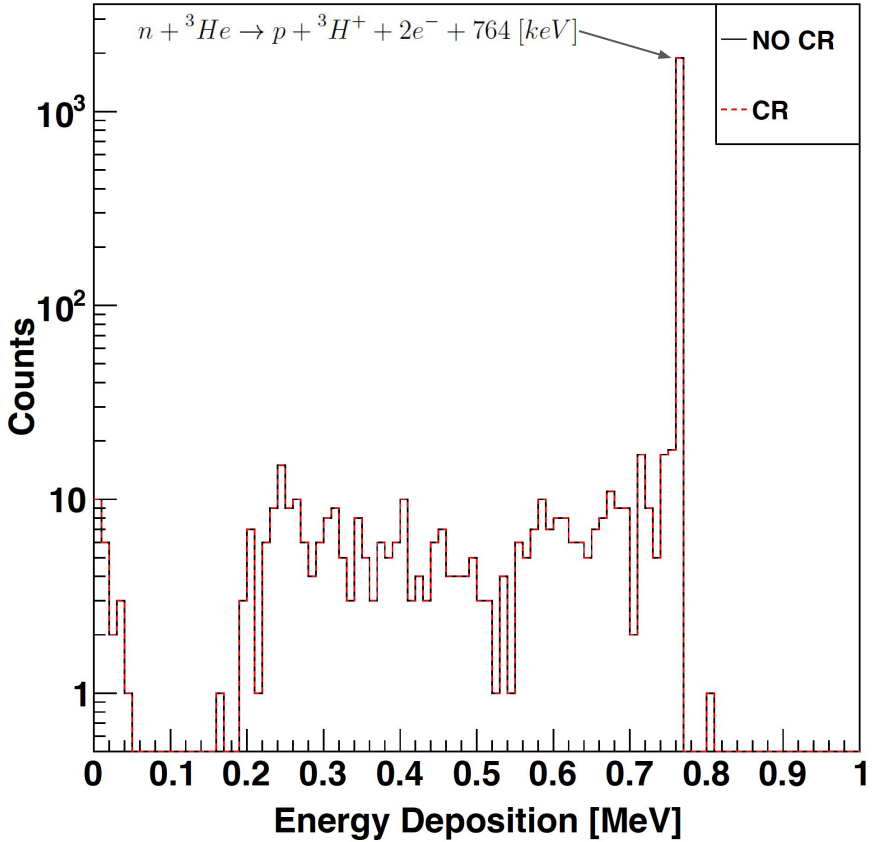


Figure 4.6.6: ***Left*:** The SolidWorks drawing for the SDSMT neutron measurement setup with the neutron testbed inside the aluminum support profiled acrylic enclosure. The LZ YBe photoneutron source with tungsten shielding is placed on the table inside the enclosure. The SDSMT neutron testbed, HDPE moderator thickness, and the ^{3}He proportional counters will go under the table. ***Middle*:** Geant4 (v10.7) simulated geometry for the setup. All the components are implemented in the simulation except the aluminum support profile frame for the support of the enclosure and the legs of the table. ***Right*:** Zoomed in Geant4 simulated geometry for the LZ YBe setup. The different components in the picture are YBe tungsten shielding (brown), beryllium volume (yellow) with ^{88}Y-disk (orange) inside it, YBe source holding table (mint), HDPE moderator (light blue) U-shaped polypropylene frame with the hole on it to support the ^{3}He tubes (sliver), the outer layer of ^{3}He proportional counter (blue) and active region (red). The dimensions and the position of all the components are described in the text.

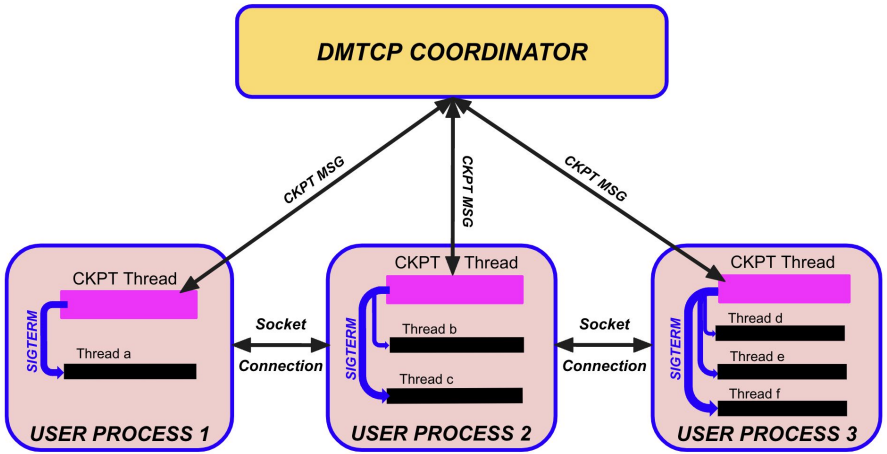
M. Timalsina, PhD Thesis



DMTCP: Simplifying Checkpoint-Restart (C/R)

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

DMTCP Architecture:

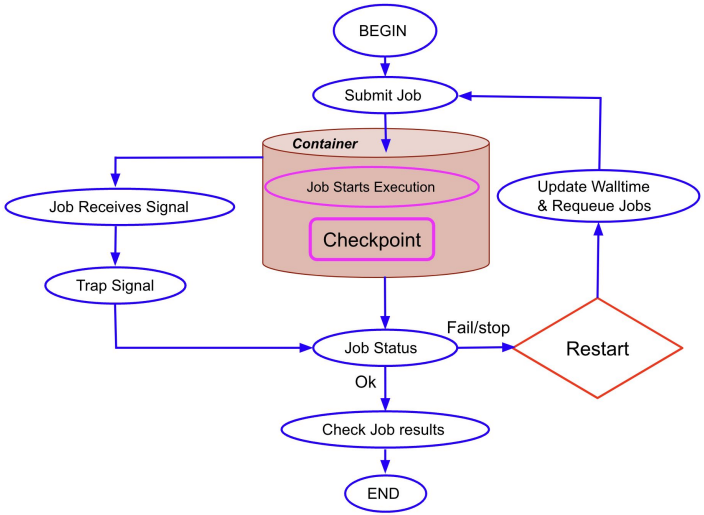


arXiv:2407.19117

- **Seamless and User-Friendly:**
  - No code or kernel modifications
  - Root access not required
  - OpenMP, Python, C/C++, Fortran, shell scripts, and resource managers like Slurm
- **Efficient and Fault-Tolerant:**
  - One coordinator per computation, multiple independent checkpoints
  - User-space operation (no need of administrative privileges)
  - runtime library and environmental variable preservation

Automated C/R Strategies using Containers

- Users submit their job scripts, with the checkpoint interval (-i), incorporating DMTCP within containers, along with necessary software packages like Geant4, CP2K
- Custom scripts (python and batch) 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



arXiv:2407.19117