Verifiable application-level checkpoint and restart framework for parallel computing

I. Gankevich I. Petriakov A. Gavrikov
D. Tereshchenko G. Mozhaiskii

Saint Petersburg State University

GRID'21

Motivation

Some claims about checkpoints that we heard:

- *"System-level checkpoints do not scale for the large number of nodes!"*
- *"Application-level checkpoints take too much effort to implement!"*

Our goals:

- Test these claims.
- Write a library for *reliable* application-level checkpoints along the way.

Benchmarks

- Benchmarks: NAS Parallel Benchmarks, miniFE.
- Programming effort: 4 working days for writing and testing checkpointing code for *all* the benchmarks.
- System-level checkpoint: DMTCP.

DMTCP checkpoints

- Userspace alternative to Berkley Lab Checkpoint and Restart (BLCR).
- Example usage

```
# launch the coordinator
dmtcp_coordinator --exit-on-last --daemon
# launch the application
dmtcp_launch --join-coordinator --coord-host $(hostname) mpiexec ...
# restart from the checkpoint
./dmtcp_restart_script.sh
```

- Peculiarities
  - Did not work with OpenMPI in our tests (we had to use MPICH).
  - Several times during tests restart failed (hanged).
  - Restart on a different set of nodes is unreliable (but would be useful in real-world scenario).
  - Restart inside SLURM job fails if you use SLURM to launch MPI processes (i.e. if you do not use the hostfile).
  - Stopped working after the system update I did yesterday.
- Overall impression: too many corner cases are not handled properly, unreliable compared to application-level checkpoints.

MPI-Checkpoint library

```
int step_min = 0;
MPI_Checkpoint checkpoint = MPI_CHECKPOINT_NULL;
int ret = MPI_Checkpoint_restore(MPI_COMM_WORLD, &checkpoint);
if (ret == MPI_SUCCESS) {
    MPI_Checkpoint_read(checkpoint, &step_min, 1, MPI_INT);
    ... // read more variables
    MPI_Checkpoint_close(&checkpoint);
}
// the main loop
for (int step=step_min; step<=step_max; ++step) {
    ... // some application logic code
    int ret = MPI_Checkpoint_create(MPI_COMM_WORLD, &checkpoint);
    if (ret == MPI_SUCCESS) {
        MPI_Checkpoint_write(checkpoint, &step, 1, MPI_INT);
        ... // write more variables
        MPI_Checkpoint_close(&checkpoint);
    }
}
```

MPI-Checkpoint library usage example

Configuration:

```
export MPI_CHECKPOINT_CONFIG=config.tmp
cat > $MPI_CHECKPOINT_CONFIG << EOF
checkpoint-prefix = ...
checkpoint-min-interval = 30m
verbose = 1
EOF
```

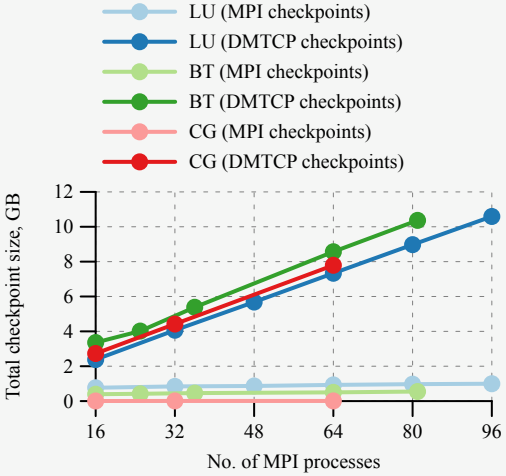
Output directory:

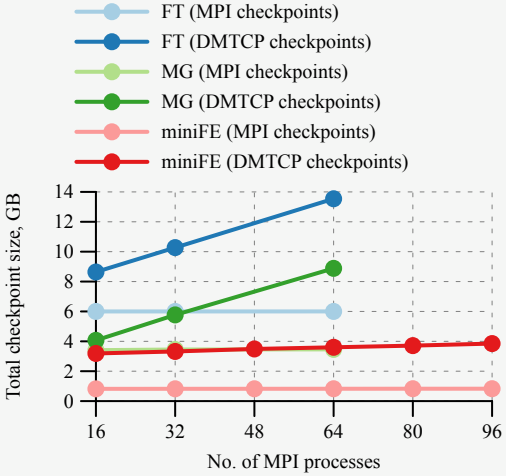
```
ls miniFE.x.1624481219.checkpoint/
0  1  10  11  2  3  4  5  6  7  8  9
```

Restart:

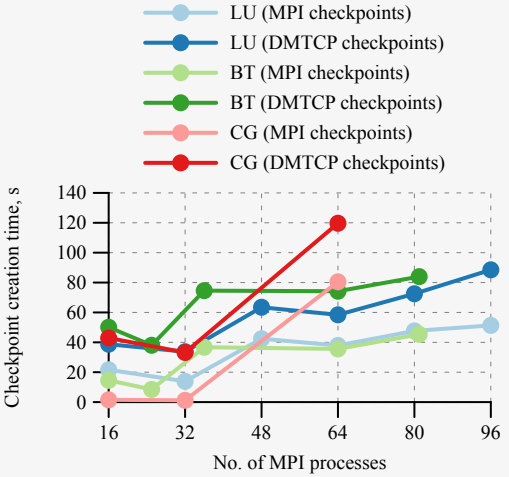
```
export MPI_CHECKPOINT=miniFE.x.1624481219.checkpoint
```

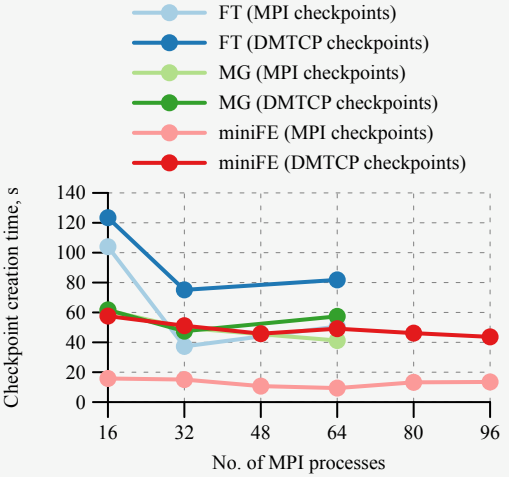
Checkpoint size





Checkpoint creation time





Other results

Checkpoint size growth rate (miniFE):

- MPI-Checkpoints: 0.2% per node (16 MPI processes)
- DMTCP checkpoints: 4% per node (16 MPI processes)

Conclusion

| *“System-level checkpoints do not scale for the large number of nodes!”*
Only if you use parallel file system. With local file system they scale.

| *“Application-level checkpoints take too much effort to implement!”*
Straightforward for applications with the main loop.

MPI-Checkpoint library (public domain):
https://github.com/igankevich/mpi-checkpoint

Cluster configuration

DMTCP	version 2.6.0, arguments: `--no-gzip`
MPICH	version 3.3.2, environment variables: `HYDRA_RMK=user`
NPB	version 3.4, class C
miniFE	version 2.0, parameters: `nx=300 ny=300 nz=300`
Compiler	GCC 7.5.0, compilation flags: `-O3 -march=native`
Cluster	6 nodes, 2 processors per node, 4 cores per processor, 2 threads per core (96 threads in total), 1 Gbit network switch
GlusterFS	version 8.0, two replicas for each file (the same nodes and network switch as the cluster)