TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

The University of Texas at Austin

System-level Transparent Checkpointing with DMTCP

PI: Gene Cooperman[1] ECSS Support: Jerome Vienne[2]

[1] Northeastern University
Boston, MA

[2] Texas Advanced Computing Center
The University of Texas at Austin
Austin, TX

September 20th, 2016

What is Checkpointing ?

Definition

Checkpoint-Restart is the ability to save a set of running processes to a checkpoint image on disk, and to later restart it from disk.

Checkpoint-Restart involves saving and restoring

- all of user-space memory
- state of all threads
- kernel state
- network state
- etc.

TACC

Why do we need to use checkpointing?

- Fault tolerance
- Scheduling and process migration
- Faster startup times
- Save/restore workspace (for interactive sessions)
- Managing “tails” (slower thread tasks) for multi-threaded applications
- Debugging
- Speculative execution (what-if scenarios)
- etc.

DMTCP: Distributed MultiThreaded CheckPointing

- Open source system-level checkpointing
- Transparent to the user
  - Works without modifying the source code or binary
- User-space
  - No kernel modules
- Handles distributed applications
  - Centralized coordinator
- Supports multiple programming language
  - C/C++, Java, Haskell, Lisp, Python, Perl, Matlab, R etc.
- Handles MPI libraries, resource managers, process managers, etc.
  - Open MPI, MVAPICH2, Intel MPI, ...

Available at: `http://dmtcp.sourceforge.net`

TACC

Table of Contents

TACC

Table of Contents

TACC

Issues with current batch sessions:

- Scheduling
  - Over-reservation of batch resources due to lack of proper estimation
  - Job termination
  - Cluster maintenance
- Debugging
  - Distributed state
  - Production (print debugging) vs. debug clusterd

Proposed Solution:

- System-level checkpointing using DMTCP

TACC

Extended batch sessions

What to do with jobs that might continue beyond maximum time of a batch slot?

- Use hooks provided in resource manager
  - Invoke checkpointing before shutdown
  - Question: How efficient is checkpointing?
  - Question: System-level versus application-level?
- Use timers when hooks not available

TACC

Checkpoint-Enabled Debugging

What to do with long-running batch jobs that crash or exhibit other problems?

- Checkpoint a long-running job prior to a problem!
  - Crash (segfault, etc.)
  - Software ”hanging”
  - Premature termination
  - Incorrect intermediate state
  - ...
- Migrate to a debug machine/cluster
  - Repeatedly restart
  - Attach a debugger
  - Interactive inspection

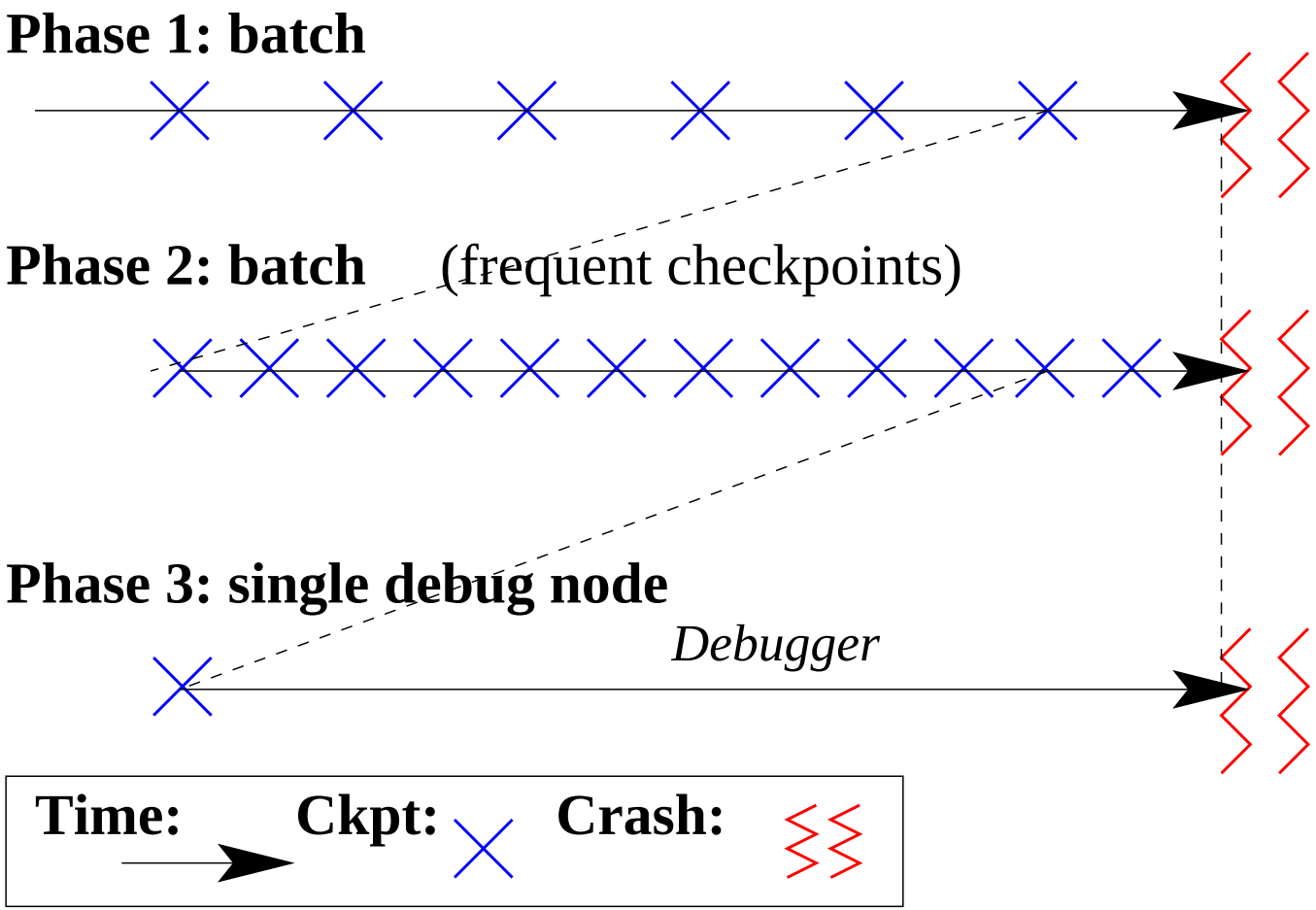
NEEDED: A better debugging experience

What to do with long-running batch jobs that crash or exhibit other problems?

- Checkpoint a long-running job prior to a problem!
- Better reproducibility
- Reduction of a problem to its essence
- Deduction
- Experimentation

TACC

Debugging: Three-phases



Debugging: Three phases [contd.]

- Submit a batch job with frequent checkpoints
  - E.g., every hour
- Restart from a checkpoint and take finer-grained checkpoints
  - E.g., every minute
- Copy all checkpoint images to a debug node/cluster and restart
  - Use your favorite debugging technique

Three-phase debugging: Version 2

Issue:

- Too many processes restarting on a single node, or a small debug cluster
  - One could suspend all but one process for a while.
  - NOT POSSIBLE: typical batch clusters do not provide virtual memory.

Solution:

- Identify a potential faulty process
- Restart just the buggy process!

Further concerns:

- Hard to predict the single buggy process out of thousands
- Lack of determinism (different buggy process on each run)

TACC

Three-phase debugging: Version 3

Issue:

- Communication-intensive applications
  - How to replay communication to the single buggy process on restart?

Solution:

- Add MPI_Send/MPI_Receive wrappers with counters
- Restart, and checkpoint after 100 counts in phase I,
- Restart, and checkpoint after 10 counts in phase II
- Restart after final MPI_Send/MPI_Receive (before the bug) in phase III.

Conclusion

- System-level transparent checkpointing is useful and practical
- Extended batch session
  - Leverage the hooks provided by resource managers to trigger checkpointing
  - Restart from checkpointed images in a different batch-queue submission
- Debugging long-running MPI applications
  - Checkpoint before the bug and restart on fewer nodes
  - Restart a single MPI process that exhibits the fault

For more details

R. Garg, J. Cao, K. Arya, G. Cooperman, and J. Vienne. ”Extended Batch Sessions and Three-Phase Debugging: Using DMTCP to Enhance the Batch Environment”. In: Proceedings of the 2016 Annual Conference on Extreme Science and Engineering Discovery Environment (XSEDE ’16). July 2016.

TACC

Table of Contents

TACC

Checkpoint-Restart with OpenSHMEM

Ali et al.(2011)

Proposed an application-specific fault tolerance mechanism

Hao et al.

- Presented at OpenSHMEM Workshop 2015
- More generic approach based on User Level Fault Tolerance (ULFM)
- Use shadow memory in which the shared memory regions of peers are backed up by peers.
- The user code is responsible for invoking a chechpoint and for restoring correct operation during a restart
- Copy the shared memory region along with privately mapped memory to a peer process during runtime $\Rightarrow$ This places added pressure on the network fabric and on the RAM.

Design modification of DMTCP to Support OpenSHMEM

The design of DMTCP had to be extended in few areas in order to support both checkpointing of modern MPI implementations and checkpointing of OpenSHEM:

- Unix domain sockets (Earlier MPI implementations generally did not use UNIX domain sockets).
- SysV shared Memory objects: Only BSD-style shared memory regions (using mmap and “MAP_SHARED”) was initially supported.
  - OpenSHMEM requires support for large shared memory regions created by the user’s application.
  - Absence of virtual memory. (Alternative strategy was created).

TACC

Experimental Setup 1/3

Stampede Cluster

- Ranked #12 on latest TOP500
- CentOS 6.4
- 6,400 nodes
  - Linux Kernel 2.6.32-431-el6
  - 16-cores (dual sockets) Sandy Bridge Xeon E5-2680
  - At least 1 Xeon Phi (KNC)
  - 32 GB RAM
- InfiniBand FDR interconnect
- SLURM resource manager
- No swapfile
- Lustre Filesystem 2.5.5

TACC

Experimental Setup 2/3

Software used:

- Intel Compiler 13.0.2.146
- MVAPICH2-X 2.0b
- NAS Benchmarks (BT & SP) with OpenSHMEM support

TACC

Experimental Setup 3/3

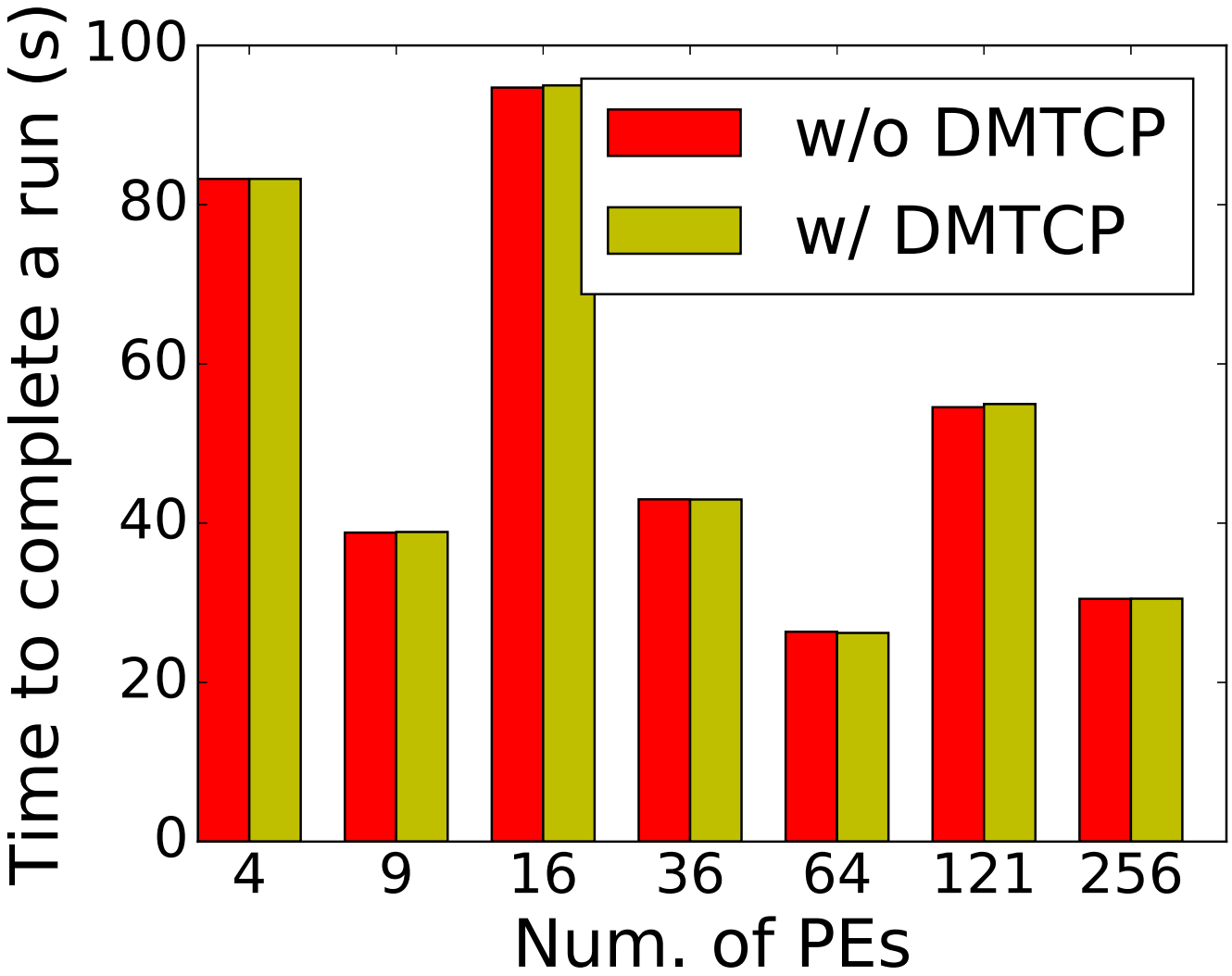
Distribution of Processes

Num of PE's	Num of Nodes	Processes per node	NAS class used
4	2	2	A
9	3	3	A
16	4	4	B
36	6	6	B
64	8	8	B
121	11	11	C
256	16	16	C

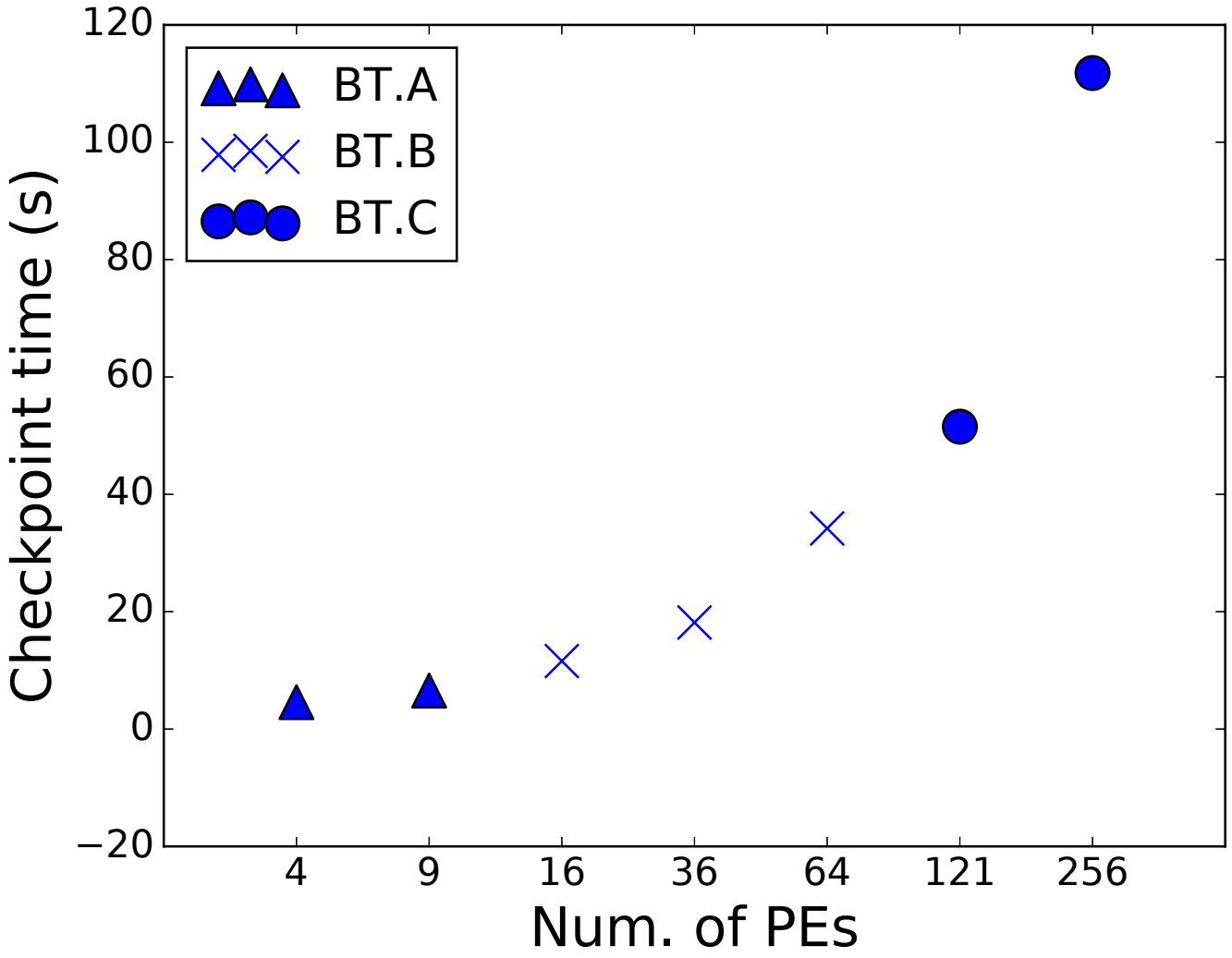
For a given number of PE's, all the runs (with and without DMTCP) were conducted on the same set of nodes to reduce the variability due to network topology and traffic.

TACC

Runtime Overhead (NAS BT)

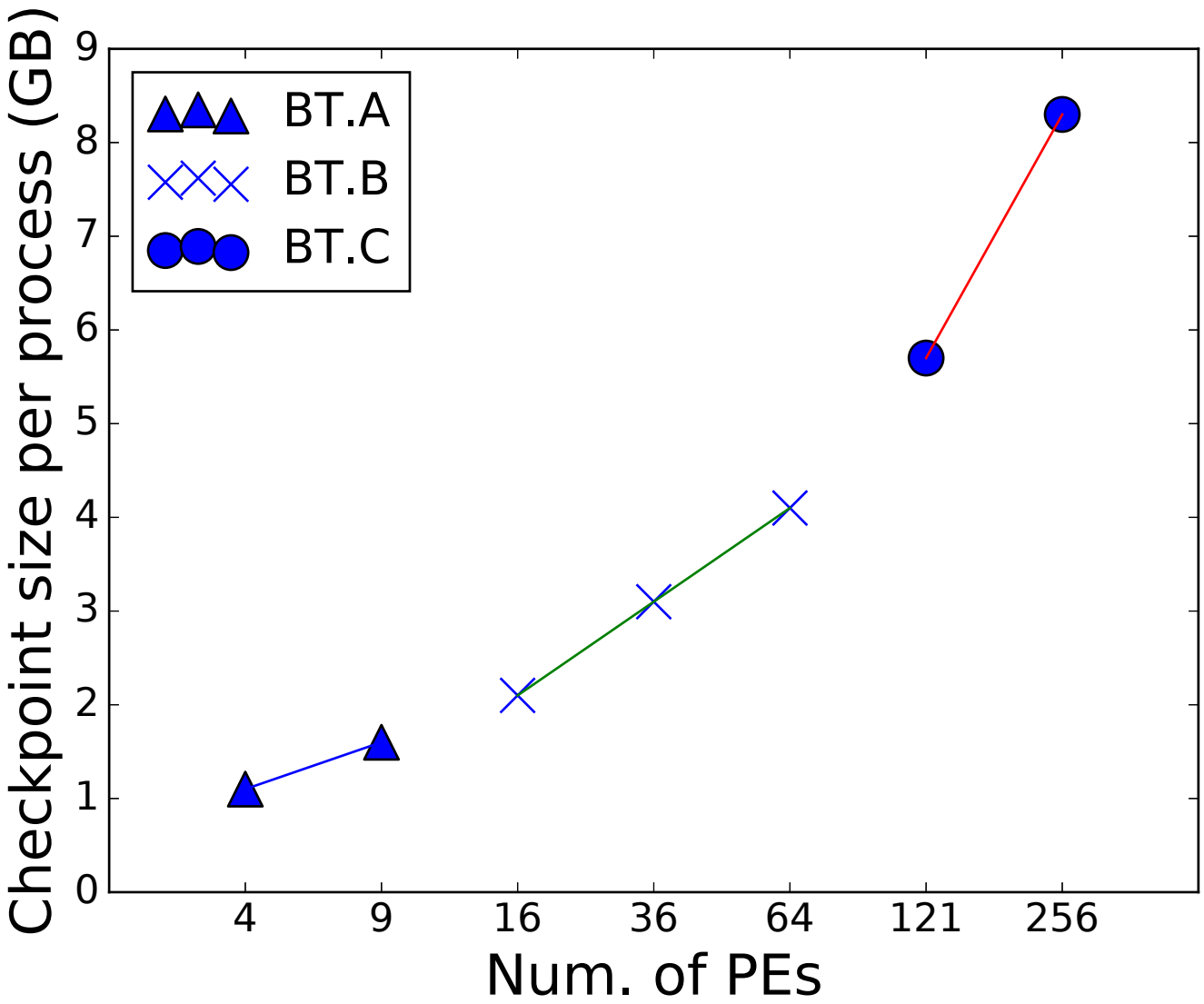


Checkpoint Times for NAS BT



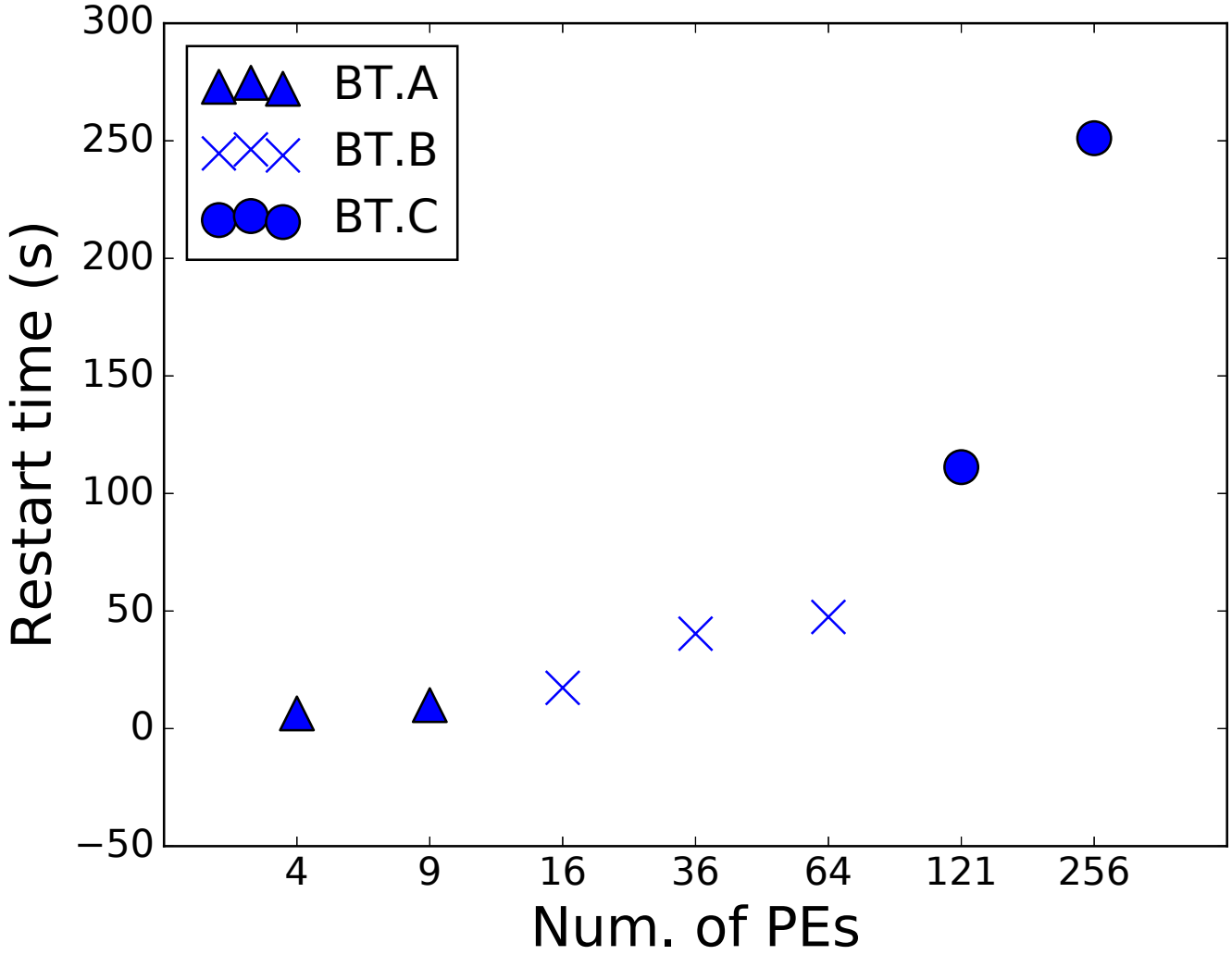
TACC

Uncompressed Image Sizes for NAS BT



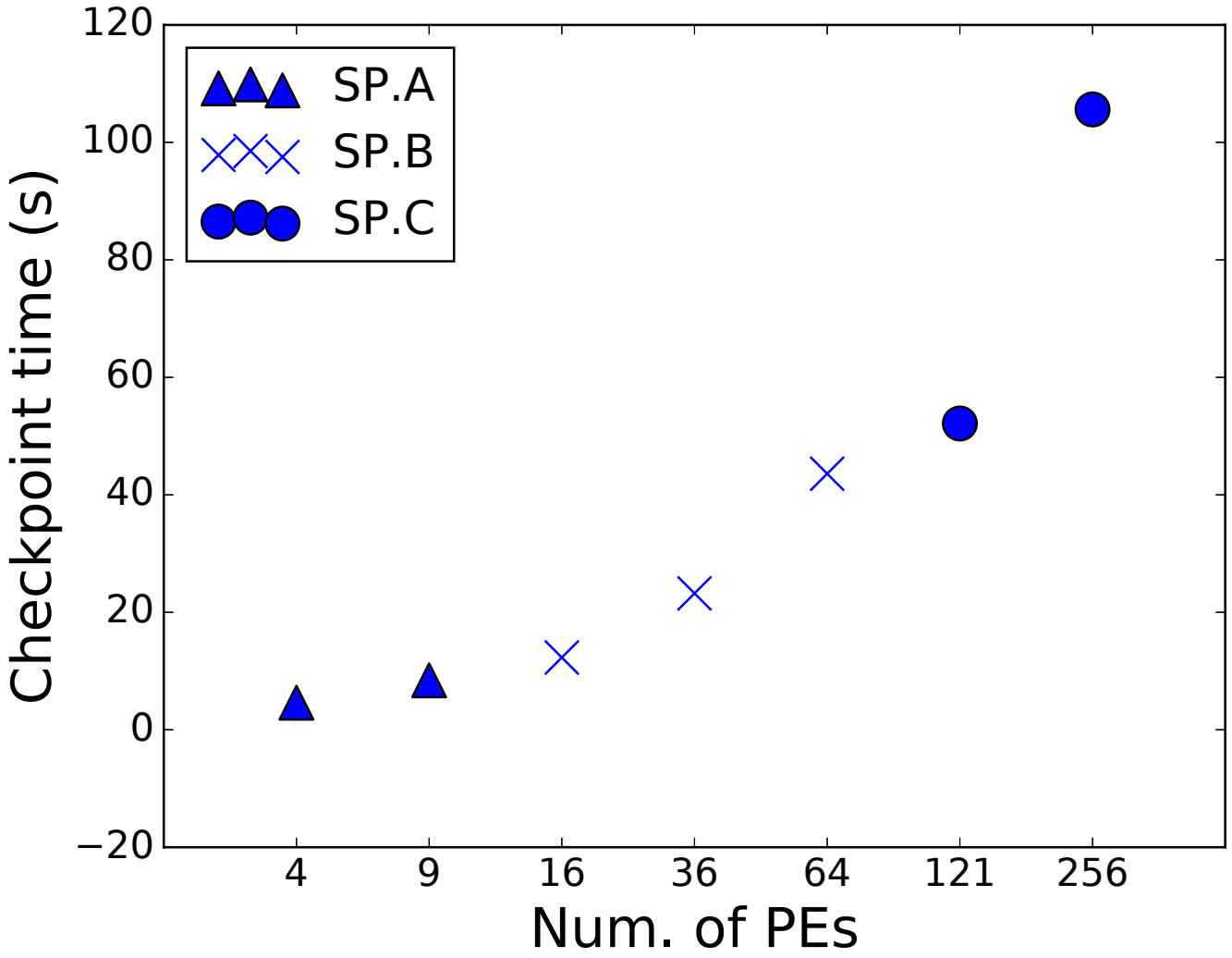
TACC

Restart times for NAS BT



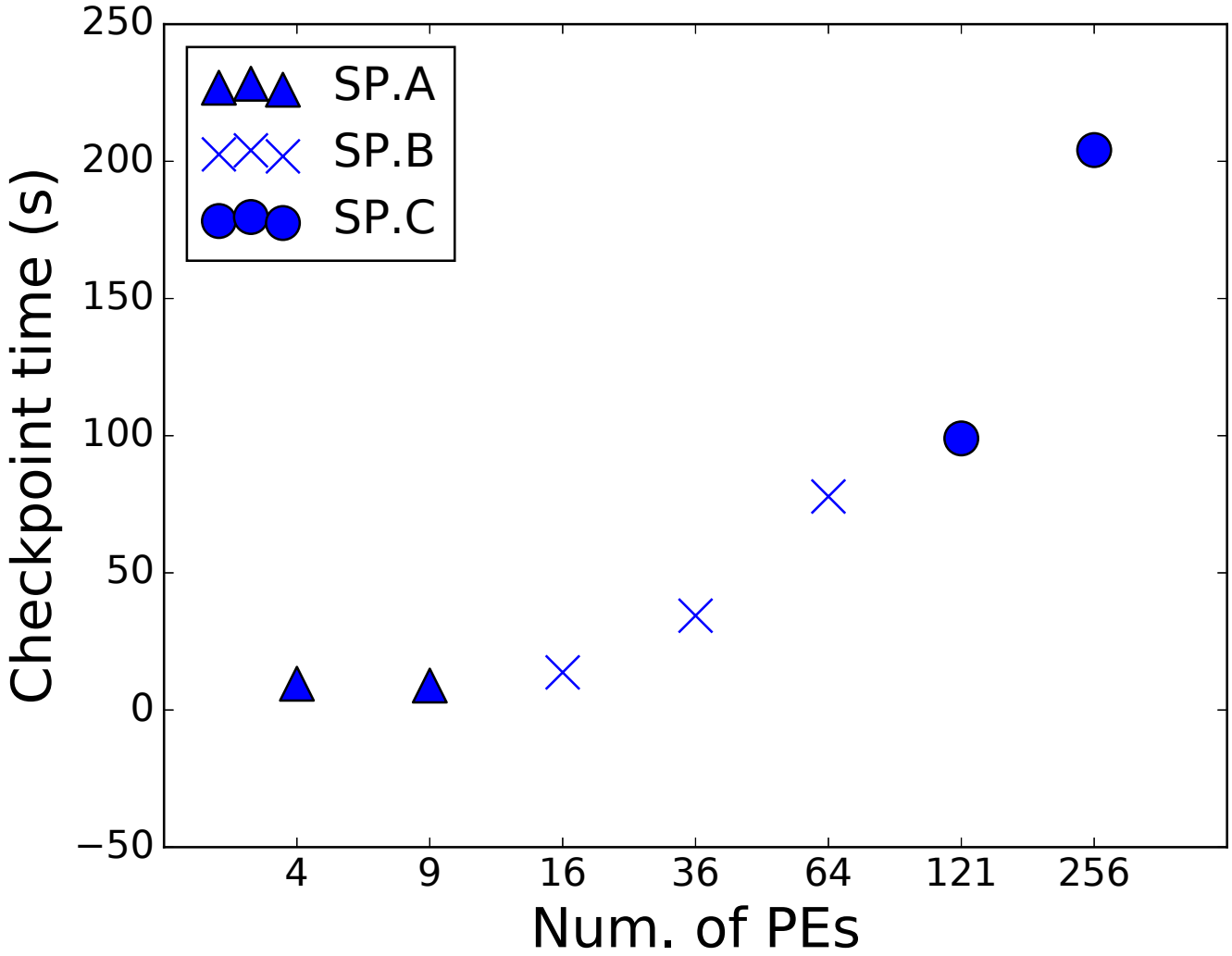
TACC

Checkpoint Times for NAS SP



TACC

Restart times for NAS SP



TACC

General comments

- At the largest scale, 256 processes, the total data written to the disk is 2.2 TB, with an effective bandwidth of 20 GB per second.
- In all the cases, the checkpoint times are dominated by the time to write the checkpoint data to stable storage, and the cost for checkpointing the state of the application is negligible.
- We observe that the largest component in a checkpoint image is an OpenSHMEM shared-memory region (90-97% of the total image size in these experiments).

TACC

Conclusion

- A system-level approach to checkpoint OpenSHMEM was presented
  - Capability of saving the state of an entire computation for restart
- Working on a leader election strategy
  - Only one copy of each shared memory region will be saved on a single node
  - Will reduce the time to write to back-end storage.

TACC

For more details

R. Garg, J. Vienne and Gene Cooperman. Scalable System-level Transparent Checkpointing for OpenSHMEM. In OpenSHMEM 2015: Third workshop on OpenSHMEM and Related Technologies. August 2016.

TACC

Table of Contents

TACC

MPI Scalability issues

- Reliable Connection mode requires a huge amount of memory at scale, for 24,000 MPI processes up to 576 millions of connections could be required => HUGE amount of memory required
- Unreliable Datagram (UD) is generally used at large scale
- How to save or replay ”in flight data” that is present in the InfiniBand network at the time of the checkpoint ?
  - How to enable transparent checkpointing to support the InfiniBand UD ?

TACC

HPCG

Num. of processes	Checkpoint time (s)	Restart time (s)	Total ckpt size (TB)	Write (ckpt) bandwidth (GB/s)
8192	136.1	215.3	9.4	69
16368	367.4	706.6	19	52
24000	634.8	1183.8	29	46

Table: Checkpoint and restart trends for HPCG; checkpoint image size for each process is 1.2 GB, with 16 images generated on each computer node.

TACC

NAMD

Num. of processes	Checkpoint time (s)	Restart time (s)	Total ckpt size (TB)	Ckpt (write) bandwidth (GB/s)
8192	41.4	111.4	2.1	51
16368	157.9	689.8	9.8	62

Table: Checkpoint and restart trends for NAMD with 8K and 16K cores (one MPI process per core); checkpoint image size for each process is 260 MB and 615 MB, respectively, with one checkpoint image per process.

TACC

NAS LU.E

Num. of processes	Runtime (s) (natively)	Runtime (s) (w/ checkpointing support)	Overhead (%)
512	596.6	601.4	0.8
1024	316.2	317.8	0.5
2048	197.6	201.9	2.2
4096	144.0	144.1	0.1

Table: Runtime overhead for NAS benchmark LU.E (class E): Times are native (without checkpointing support) and with checkpointing support.

TACC

NAS LU.E: Chkpt/Restart Trend

Num. of processes	Checkpoint time (s)	Restart time (s)	Checkpoint size per process (MB)
1024	14.5	15.8	428
2048	24.2	20.6	342
4096	33.7	36.9	300
8192	65.8	107.6	280
16368	131.8	514.7	285

Table: Checkpoint and restart trends for LU.E

NAS LU.E: Different MPI libraries

MPI implementation	Checkpoint time (s)	Restart time (s)	Checkpoint size per process (MB)
Intel MPI	298.9	191.8	775
Open MPI	299.7	128.5	520

Table: Comparison of two MPI implementations; LU.E for 500 processes

TACC

For more details

J. Cao, K. Arya, R. Garg, S. Matott, D.K. Panda, H. Subramoni, J. Vienne and G. Cooperman. System-level Scalable Checkpoint-Restart for Petascale Computing. In IEEE ICPADS, Wuhan, China, December 2016.

TACC

Table of Contents

TACC

Conclusion

- DMTCP: an interesting project
- Multiples issues/Solutions found during this ECSS project
  - Debugging...
  - OpenSHMEM
  - UD support

Future Work

- Support of Omni-Path
  - Currently working on support of PSM
- Working on a leader election strategy
  - Only one copy of each shared memory region will be saved on a single node
  - Will reduce the time to write to back-end storage.