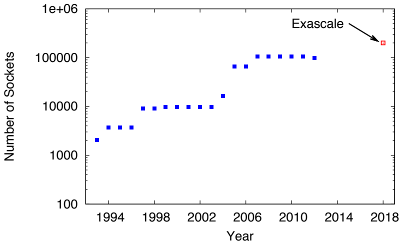
Scalable In-memory Checkpoint with Automatic Restart on Failures

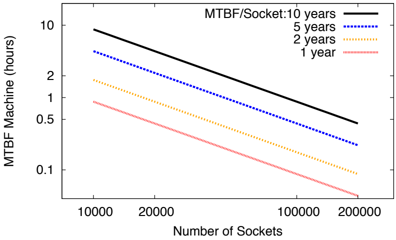
Xiang Ni, Esteban Meneses, Laxmikant V. Kalé

Parallel Programming Laboratory
University of Illinois at Urbana-Champaign

November, 2012
8th Joint Lab Workshop

Problem





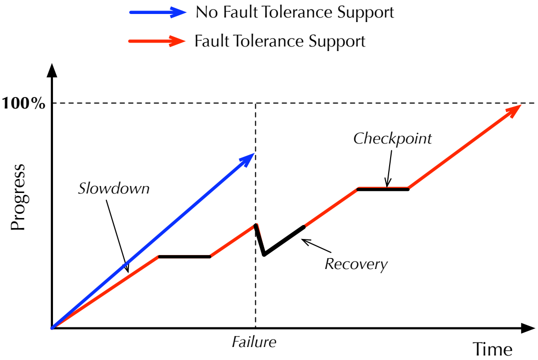
- Increasing number of sockets
  - Sequoia 98304, predicted Exascale 200000
- More frequent failures
  - MTBF of the Exacale machine will be **720 seconds** if MTBF per socket remains at 5 years.

Our Philosophy

- Runtime system support for fault tolerance
  - Checkpoint Restart
  - Message Logging
  - Proactive Migration

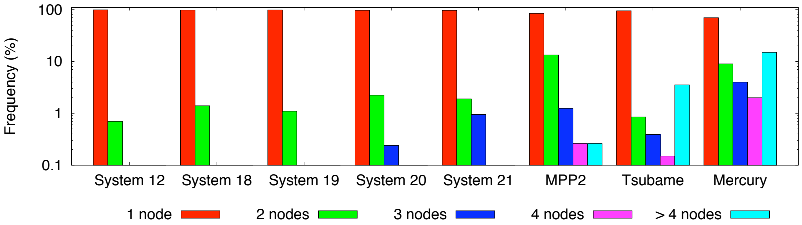
Our Philosophy

- Runtime system support for fault tolerance
- Keep progress rate despite of failures
  - Optimize for the common case
  - Minimize performance overhead



Optimize for the common case

- Failures rarely bring down more than one node
- In Jaguar (now Titan, top 1 supercomputer), 92.27% of failures are individual node crashes

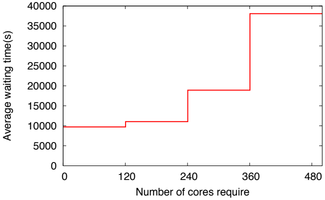


Minimize performance overhead

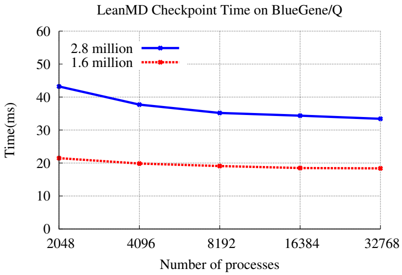
- Decrease interference with application
- Parallel recovery
- Automatic restart:
  - Failure detection in runtime system
  - Immediate rollback-recovery
- Faster checkpoint

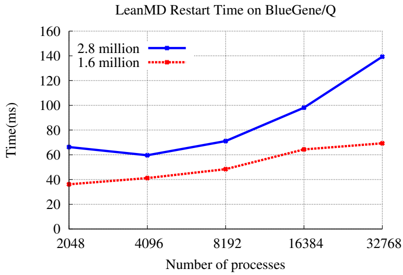
Minimize performance overhead

- Decrease interference with application
- Parallel recovery
- Automatic restart:
  - Failure detection in runtime system
  - Immediate rollback-recovery
- Faster checkpoint

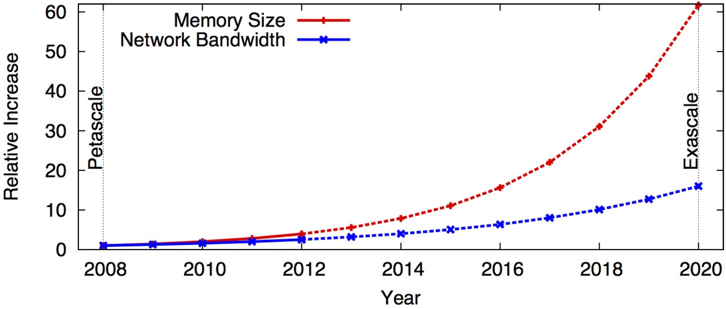


Checkpoint and Restart for Leanmd





Limitation of Checkpoint/Restart



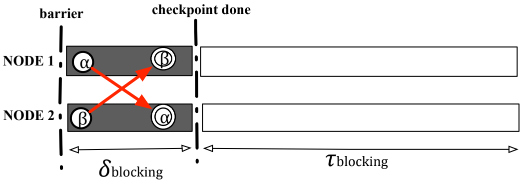
- Increase in memory size per year: 41%
- Increase in network bandwidth per year: 26%

Outline

Outline

1. Checkpoint/Restart
   - Synchronous
   - Asynchronous
   - Model
2. Minimize checkpoint interference to application
   - Priority sending queue
   - Opportunistic vs. random scheduling
3. Relieve memory pressure with SSD
4. Experiments
5. Conclusion

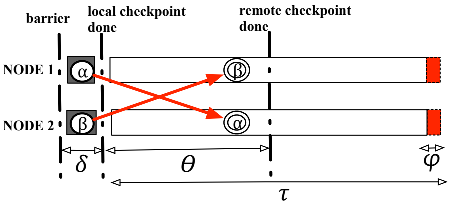
Synchronous Checkpoint



$\tau_{blocking}$	checkpoint interval
$\delta_{blocking}$	checkpoint overhead

- Each node has a buddy node to store the checkpoint.
- Resume computation after all the nodes have successfully saved the checkpoints in their buddy nodes.

Solution: Asynchronous Checkpoint



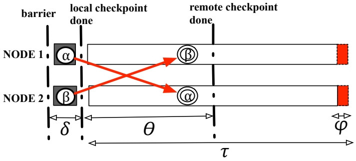
τ	checkpoint interval
δ	local checkpoint overhead
θ	overlap period
φ	remote checkpoint interference

- Resume computation as soon as each node stores its own checkpoint (local checkpoint).
- Interleave the transmission of the checkpoint to buddy with application execution (remote checkpoint).

Drawback

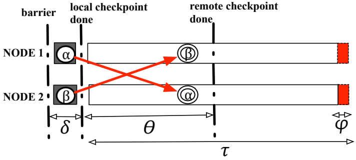
- Probability to roll back to the previous checkpoint when checkpointing overlaps with application
- Interference of the remote checkpoint to application

$$T = T_s + T_{local} + T_{overhead} + T_{rework} + T_{restart}$$



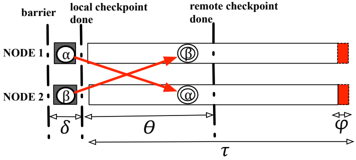
- T_s Workload
- T_{local} Time for local checkpoint
- $T_{overhead}$ Interference of global checkpoint
- T_{rework} Lost work for application
- $T_{restart}$ Time to restart application

Local Checkpoint Overhead



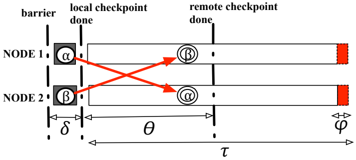
- Local checkpoint T_{local}
  - Checkpoint interval τ
  - Work finished in one checkpoint interval $\tau - \varphi$
  - Number of checkpoints $\frac{T_s}{\tau - \varphi}$

Local Checkpoint Overhead



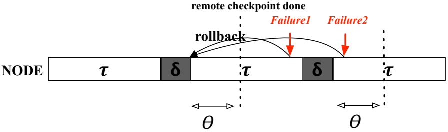
- Local checkpoint T_{local}
  - Checkpoint interval τ
  - Work finished in one checkpoint interval $\tau - \varphi$
  - Number of checkpoints $\frac{T_s}{\tau - \varphi}$
  - Local checkpoint overhead for one checkpoint δ
  - $T_{local} = \frac{T_s}{\tau - \varphi}\delta$

Remote Checkpoint Interference



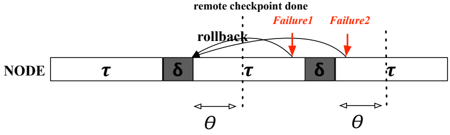
- Interference of remote checkpoint to application $T_{overhead}$
  - Interference for one checkpoint φ
  - $T_{overhead} = \frac{T_s}{\tau - \varphi} \varphi$

Rework and Restart Time



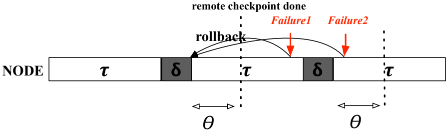
- Rework time T_{rework}
    - Number of failures $\frac{T}{MTBF}$

Rework and Restart Time



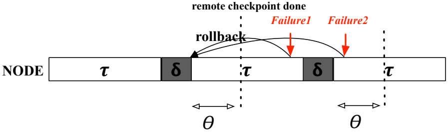
- Rework time T_{rework}
  - Number of failures $\frac{T}{MTBF}$
  - Failure 1: at least $\theta - \varphi$

Rework and Restart Time



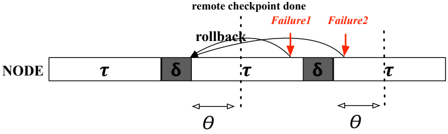
- Rework time T_{rework}
  - Number of failures $\frac{T}{MTBF}$
  - Failure 1: at least $\theta - \varphi$
  - Failure 2: at most $\theta - \varphi + \tau + \delta$

Rework and Restart Time



- Rework time T_{rework}
  - Number of failures $\frac{T}{MTBF}$
  - Failure 1: at least $\theta - \varphi$
  - Failure 2: at most $\theta - \varphi + \tau + \delta$
  - $T_{rework} = \frac{T}{MTBF}(\frac{\tau+\delta}{2} + \theta - \varphi)$

Rework and Restart Time



- Rework time T_{rework}
  - Number of failures $\frac{T}{MTBF}$
  - Failure 1: at least $\theta - \varphi$
  - Failure 2: at most $\theta - \varphi + \tau + \delta$
  - $T_{rework} = \frac{T}{MTBF}(\frac{\tau+\delta}{2} + \theta - \varphi)$
- Restart time $T_{restart}$
  - Restart time for one failure R
  - $T_{restart} = \frac{T}{MTBF} R$

$$T = T_s + \frac{T_s}{\tau - \varphi}\delta + \frac{T_s}{\tau - \varphi}\varphi + \frac{T}{MTBF}\left(R + \frac{\tau + \delta}{2} + \theta - \varphi\right)$$

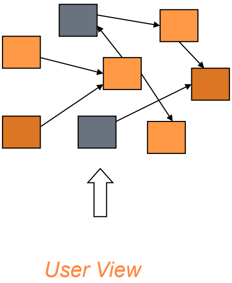
$$T_{blocking} = T_s + \frac{T_s}{\tau_{blocking}}\delta_{blocking} + \frac{T_{blocking}}{MTBF}\left(R + \frac{\tau_{blocking} + \delta_{blocking}}{2}\right)$$

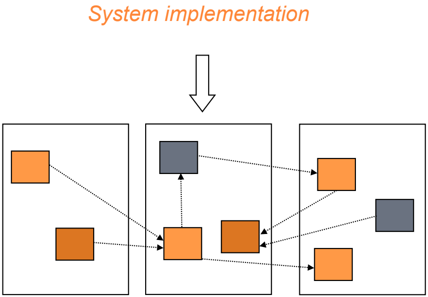
$$Benefit = \frac{T_{blocking} - T}{T_{blocking}}$$

Outline

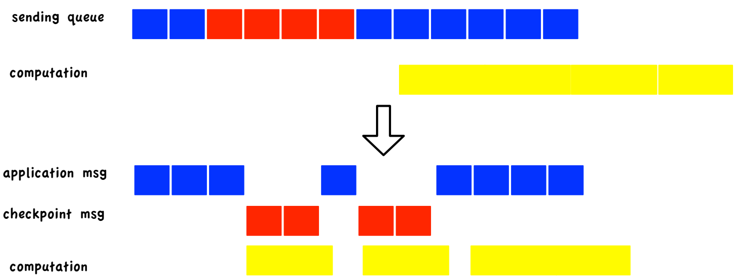
Charm++ Runtime System

- Object based over-decomposition
- Asynchronous method invocation
- Migratable-object runtime system
- Worker thread & Communication thread

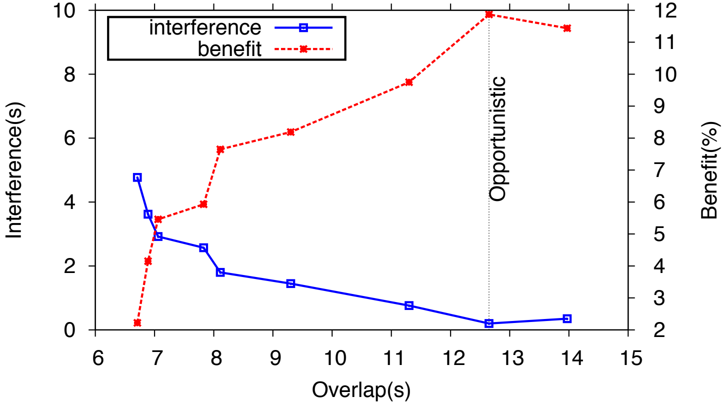




Minimize Checkpoint Interference



- Separate checkpoint message queue
- Send checkpoint message only when there is no application message ready to be sent
- Better overlap with computation



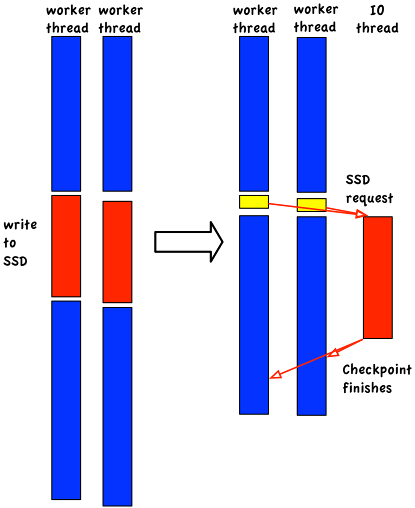
- Use lottery scheduling to change the overlap period
- Probabilistic deciding whether application or checkpoint queue can send message

Outline

Choose Data to Store in SSD

- Solid State Drive: becoming increasingly available on individual nodes
- Full SSD strategy
- Half SSD strategy
  - Only store remote checkpoint in SSD
  - Faster checkpoint and restart

Asynchronous Checkpointing to SSD with IO thread



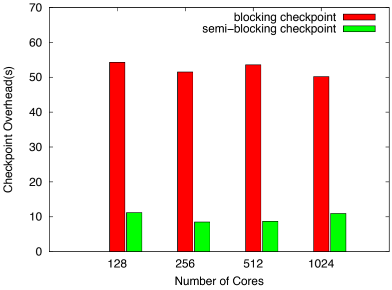
- IO threads
  - *Write checkpoint to/Read checkpoint from* SSD
    When receive request from worker thread.
  - Notify worker thread
    When SSD is done with certain request.

Outline

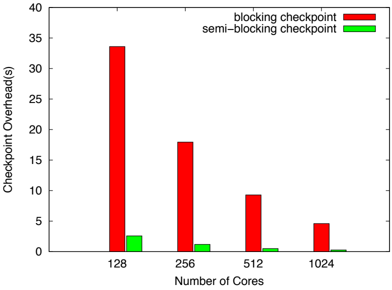
Machine

- Trestles @ SDSC
- 324 32-core nodes
- 120 GB flash memory (SSD) per node
- 100 Teraflops
- Applications
  - Wave2D: stencil computation
  - ChaNGa: N-Body simulation

Single Checkpoint Overhead



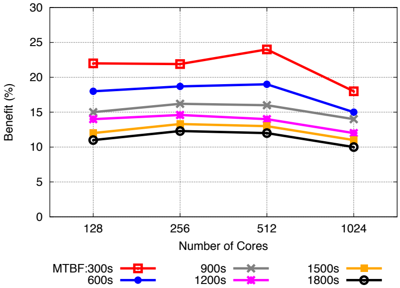
Wave2D Weak Scale



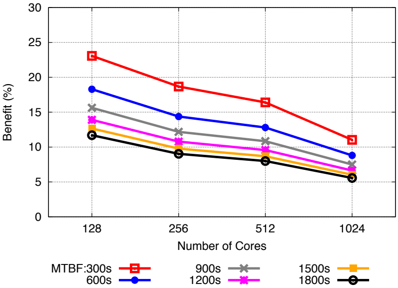
ChaNGa Strong Scale

- Semi-Blocking checkpoint reduces checkpoint overhead significantly.

Semi-Blocking Benefit



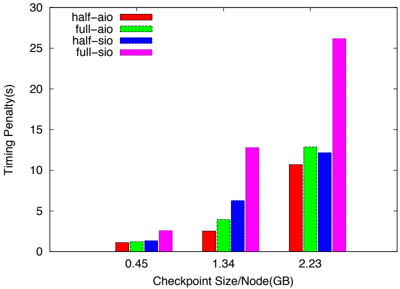
Wave2D Weak Scale

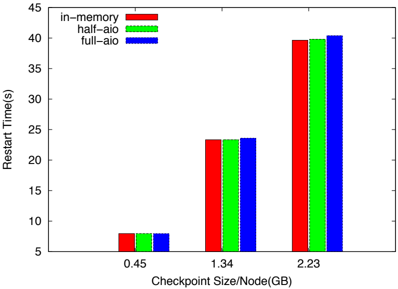


ChaNGa Strong Scale

- Semi-Blocking checkpoint reduces the total execution time up to 22%.

Checkpoint/Restart on SSD





- Half SSD strategy with asynchronous IO reduces the timing penalty for checkpointing to SSD
- Restart from SSD does not incur extra overhead

aio	asynchronous IO
sio	synchronous IO

Outline

Conclusion

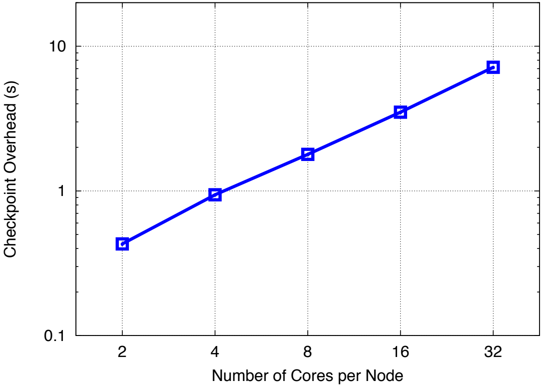
- Asynchronous checkpointing can help hide the checkpoint overhead.
- SSD can be used in checkpointing to relieve memory pressure with little overhead.

Future Work

- Log analysis is very helpful
  - Failure distributions
  - Cluster usage
- Failure prediction with different fault tolerate actions
  - Proactive migration
  - Proactive checkpoint
- Multilevel checkpointing

Thank you!

Increasing Checkpoint Overhead



- Checkpoint size: 16MB per core
- Checkpoint data is sent to another node across the network