A feasibility study of checkpoint/restart as a fault tolerance technique

Faisal Shahzad

16.05.2014

Challenge

- Nowadays, the increasing computational capacity is mainly due to extreme level of hardware parallelism.
- The reliability of hardware components does not increase with the similar rate.
- With future machines, the Mean time to failure is expected to be in minutes and hours.
- Absence of fault tolerant environment will put precious data at risk.

Fault Tolerance

- **Faults:**
  - Hardware failures (processor, memory, power supply or network etc.)

  →Normal programs abort

- **Fault Tolerance:**
  - A property that guarantees the normal program execution either by resisting or recovering from faults

  → Support required on application and/or operating system level

Fault Tolerance Approaches

1. **Algorithm Based Fault Tolerance (ABFT)**
2. **Message Logging**
3. **Redundancy**
4. **Fault Prediction (*proactive fault tolerance*)**
5. **Checkpoint/Restart (C/R)**

Each of these fault tolerance approaches carries overhead in terms of time and/or resources

Checkpoint/Restart optimizations

1. Application level checkpointing
   - Minimal checkpoint data

2. **Asynchronous checkpointing**

3. **Multi-level checkpointing**

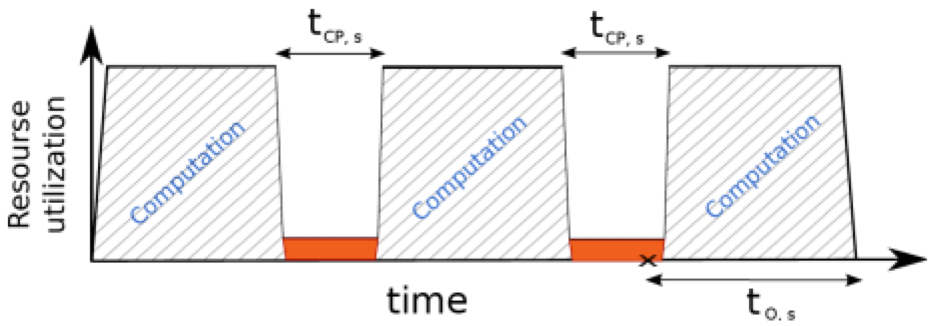
4. Checkpoint compression

5. …

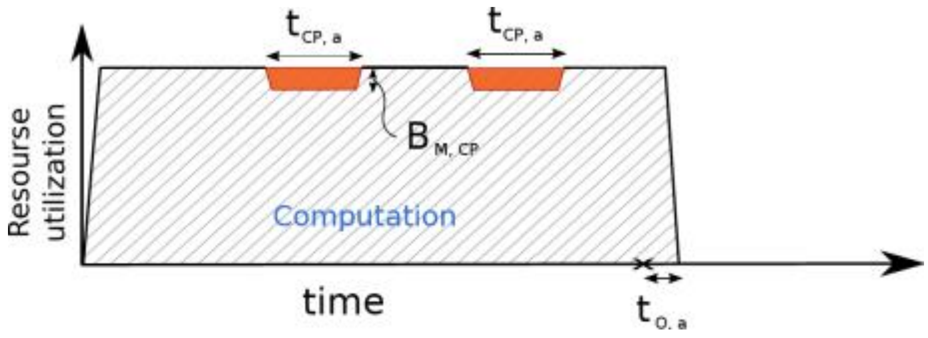
ASYNCHRONOUS CHECKPOINTING

Synchronous vs. asynchronous checkpointing

- Synchronous checkpointing:
  - Computation halts for I/O time
  - High execution time overhead

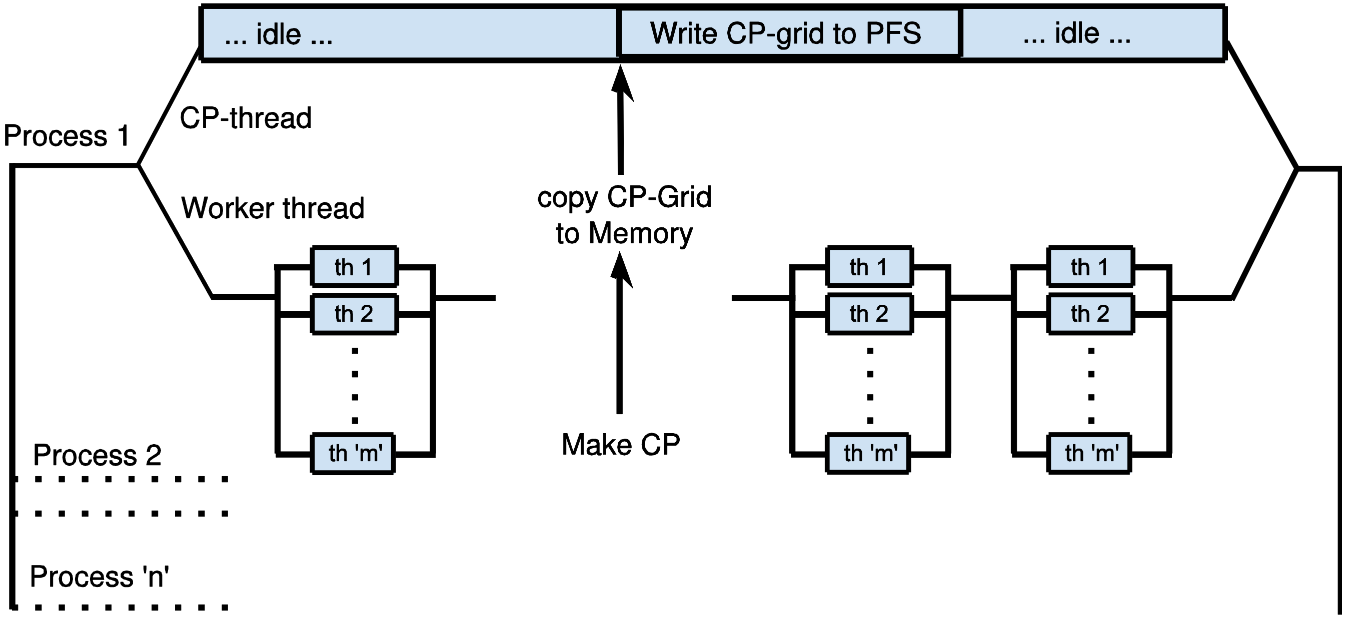


- Asynchronous checkpointing:
  - Using dedicated threads for performing asynchronous I/O
  - Low execution time overhead
  - An in-memory copy of checkpoint is required.



Asynchronous checkpointing by dedicated threads (I)

- Hybrid approach (with nested openmp parallelism)



- Flexible
  - 1 Checkpoint thread per core
  - 1 Checkpoint thread per socket
  - 1 Checkpoint thread per node

Experimental Framework

- Application:
  - A prototype CFD solver based on Lattice Boltzmann Method (LBM).

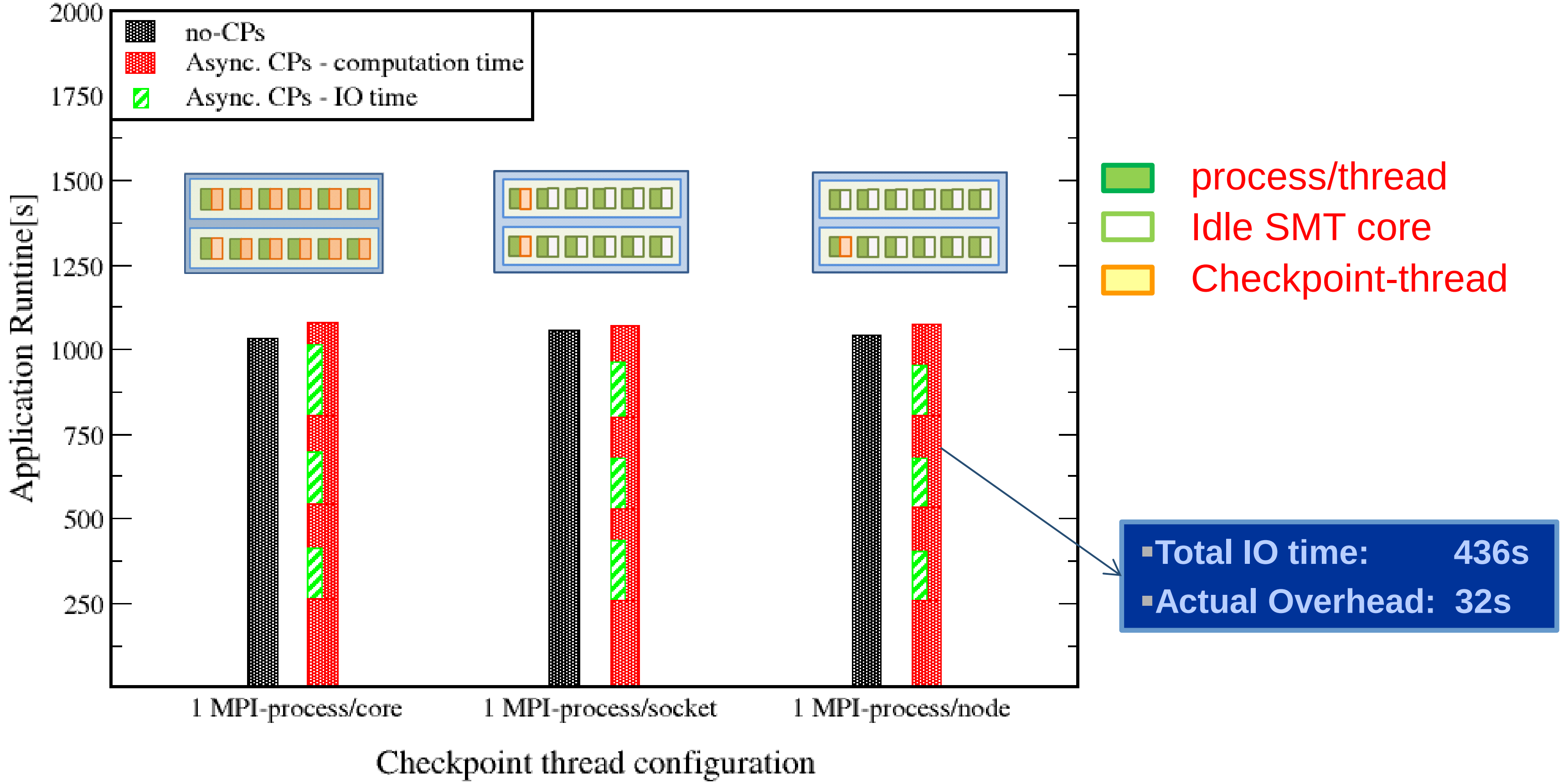
- Cluster:
  - LiMa (Erlangen) : QDR Infiniband cluster, **500** nodes (Dual socket Intel Xeon 5650 “Westmere”), Lustre based PFS Bandwidth ~ **3GB/s**

- Approaches:
  - Synchronous CP
  - Asynchronous CP

Asynchronous Checkpointing

- Hybrid (MPI-OpenMP) configuration performance comparison

Cluster: LiMa, num. of nodes = 32, PFS = LXFS, Aggregated CP size = 200 GB/CP

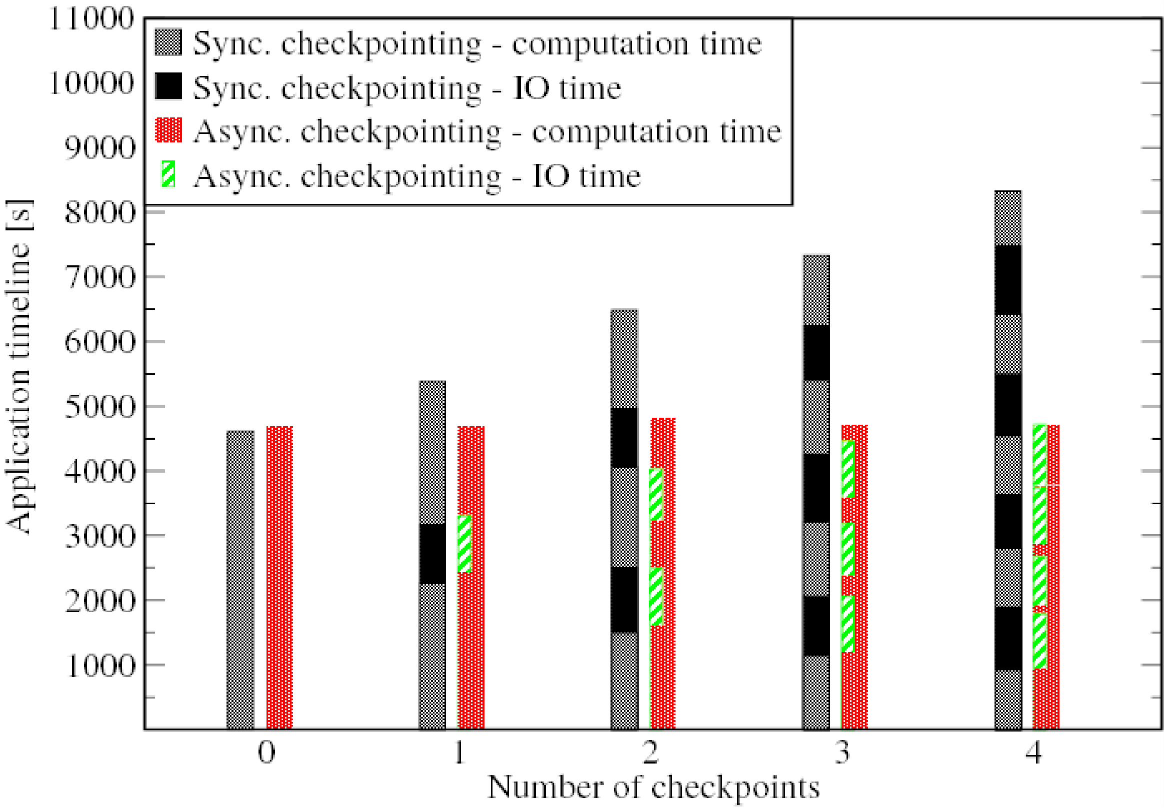


- **Total IO time: 436s**
- **Actual Overhead: 32s**

Asynchronous vs. Synchronous Checkpointing

- LiMa

Num. of nodes = 128, np = 1536, PFS = LXFS, Aggregated CP size = 800GB/CP



% overhead

- **1 Sync. CP = 20 %**
- **1 Async. CP = 0.4 %**

Asynchronous checkpointing

- Critical parameter → checkpoint frequency
  - System parameters, checkpoint latency, restart time ,…
  - Upper limit on the number of checkpoints

- Limitations
  - In-memory copy of the checkpoint data costs
    1. Extra memory space (in worst case, can be up to 50%)

    2. Time (can be avoided)

MULTI-LEVEL CHECKPOINTING

Using Scalable Checkpoint Restart (SCR) library

Scalable Checkpoint/Restart (SCR) Library

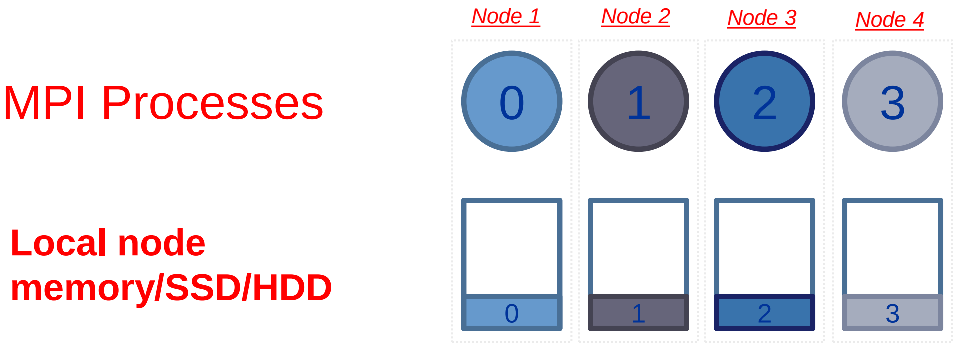
- Scalable Checkpoint/Restart is a library developed by LLNL(Adam Moody)

- Key idea
  - To store ***checkpoint data*** redundantly on ***compute nodes*** and making occasional checkpoints on the parallel file system (PFS).

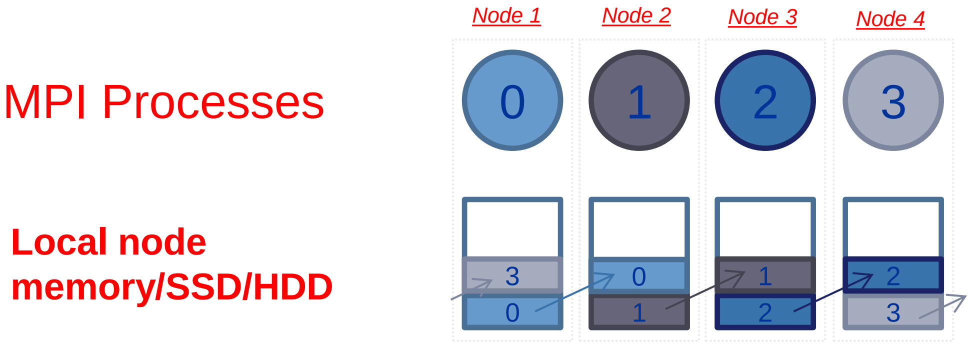
- Advantages
  - Scalable checkpointing: Every additional node adds to ***more storage space*** and ***bandwidth***
  - Scalable restart: Restart data on cluster nodes -> ***less restart time***.
  - Reduced load on PFS for making checkpoint.

SCR: Checkpointing Features (I)

- **LOCAL**



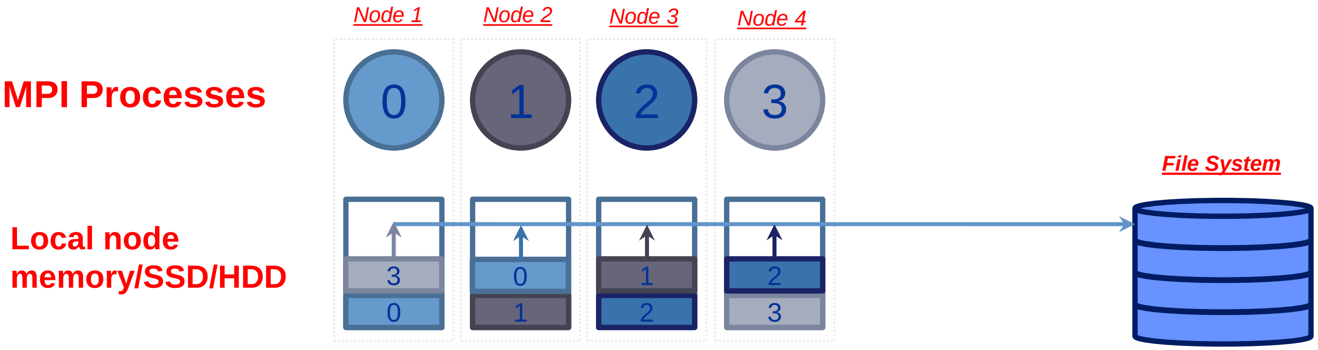
- **PARTNER**



- **PARTNER XOR: (similar to RAID5)**
  - Makes XOR checkpoints for sets of nodes

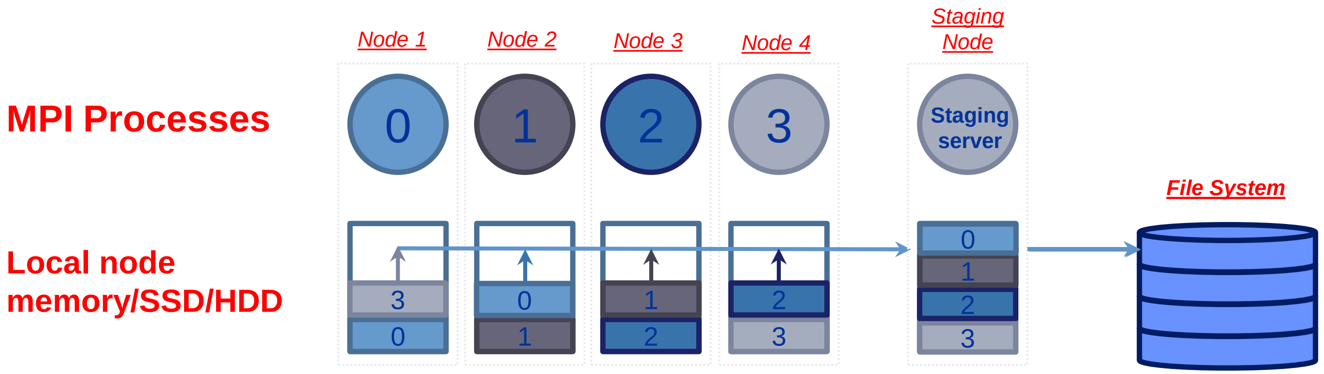
SCR: Checkpointing Features (II)

- **Parallel File System (PFS) level checkpoints**
  - In order to deal with catastrophic failures, PFS-level checkpoints can be taken.



SCR: Checkpointing Features (III)

- **Non-blocking PFS-level checkpoints**
  - PFS-level checkpoints are taken in a non-blocking way with the help of dedicated staging-nodes.



SCR: Restart Mechanism

- **Scalable Restart**
  - Restart from node, neighbor level checkpoints (if consistent checkpoint state is available)
  - If node-level consistent copy is not available for all the processes, restart is done by reading PFS level checkpoints.

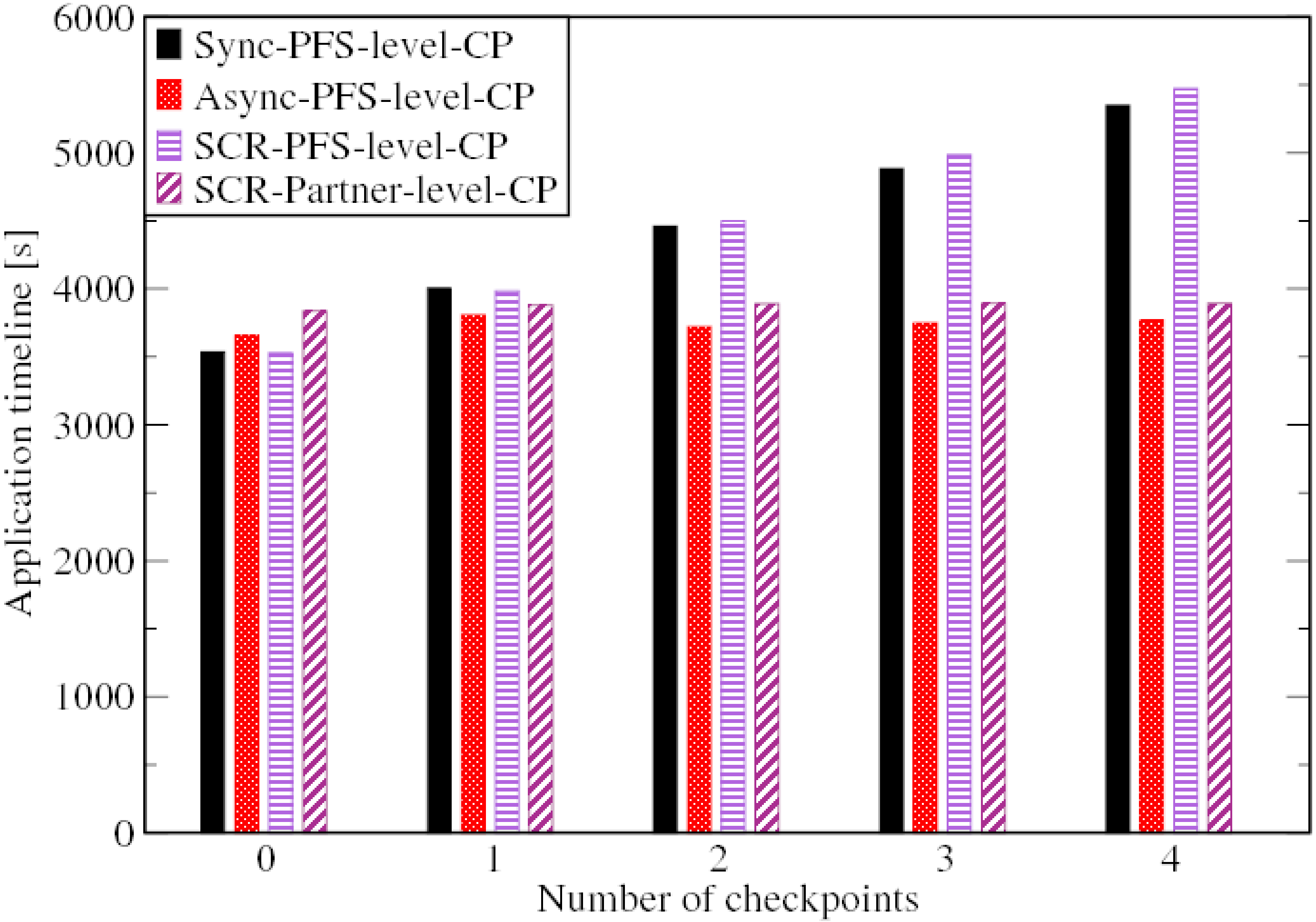
Application Requirements

- **MPI based**
- **Checkpoint mechanism**
  - SCR redirects and manages every checkpoint on node-level and PFS-level
  - Globally-coordinated checkpoint
- **Restart mechanism**
  - SCR finds the consistent copy of checkpoint that is least expansive to restart from
- **Enough memory/SSD/HDD space on nodes to store node-level checkpoints**
- **USAGE:**
  - via API calls around C/R routines
- **Limitation:**
  - Every checkpoint is treated as a complete checkpoint identity

Async. vs. Sync. vs. SCR Checkpointing

- LBM Benchmark (LiMa)

Num. of nodes = 128, PFS = LXFS, Aggregated CP size = 510 GB /CP



- **<u>% overhead:</u>**
- **1 Sync. CP = 13 %**
- **1 Async. CP = 1.3 %**
- **1 Partner. CP = 1 %**

AUTOMATIC FAULT TOLERANCE APPLICATION (AFT)

Automatic Fault Tolerance Application (AFT)

- **Automatic fault tolerance application (AFT)**
  - In the absence of failed processes, the algorithm itself is able to detect and correct the incorrectly produced results

- **FT - MPI ?**

- **GPI (Global address space Programming Interface)**
  - Fault tolerance → In case of single node failure, rest of the nodes stay up and running

<u>Message Passing Interface</u>

- „Traditionally“ single sided communication not possible
- Read/write requires both processes to acknowledge communication
- Single node crash → All nodes crash

<u>PGAS (Partitioned Global Address Space)</u>

- Read and write global data single sidedly
- Motivation -> simplicity (with scalability)
- User needs to be careful about synchronization.
- e.g. GPI (Global address space Programming Interface), GA (Global Arrays), UPC (Unified Parallel C) …

AFT: GPI Introduction

- Developed by Fraunhofer IWTM

- Based on PGAS programming model

- Two memory parts
  - Local: only local to the GPI process (and its threads)
  - Global: Available to other processes for reading and writing.

- Enables fault tolerance
  - via providing TIMEOUT for every communication call.

AFT: GPI - Application requirements

- **Algorithm based on PGAS model**

- **For effective fault tolerance**
  - No global synchronization, barriers
  - Each GPI-process communicates with certain subset of GPI-processes (e.g. neighbors)
  - In case of failures, rest of the processes detect errors in results and correct them accordingly.

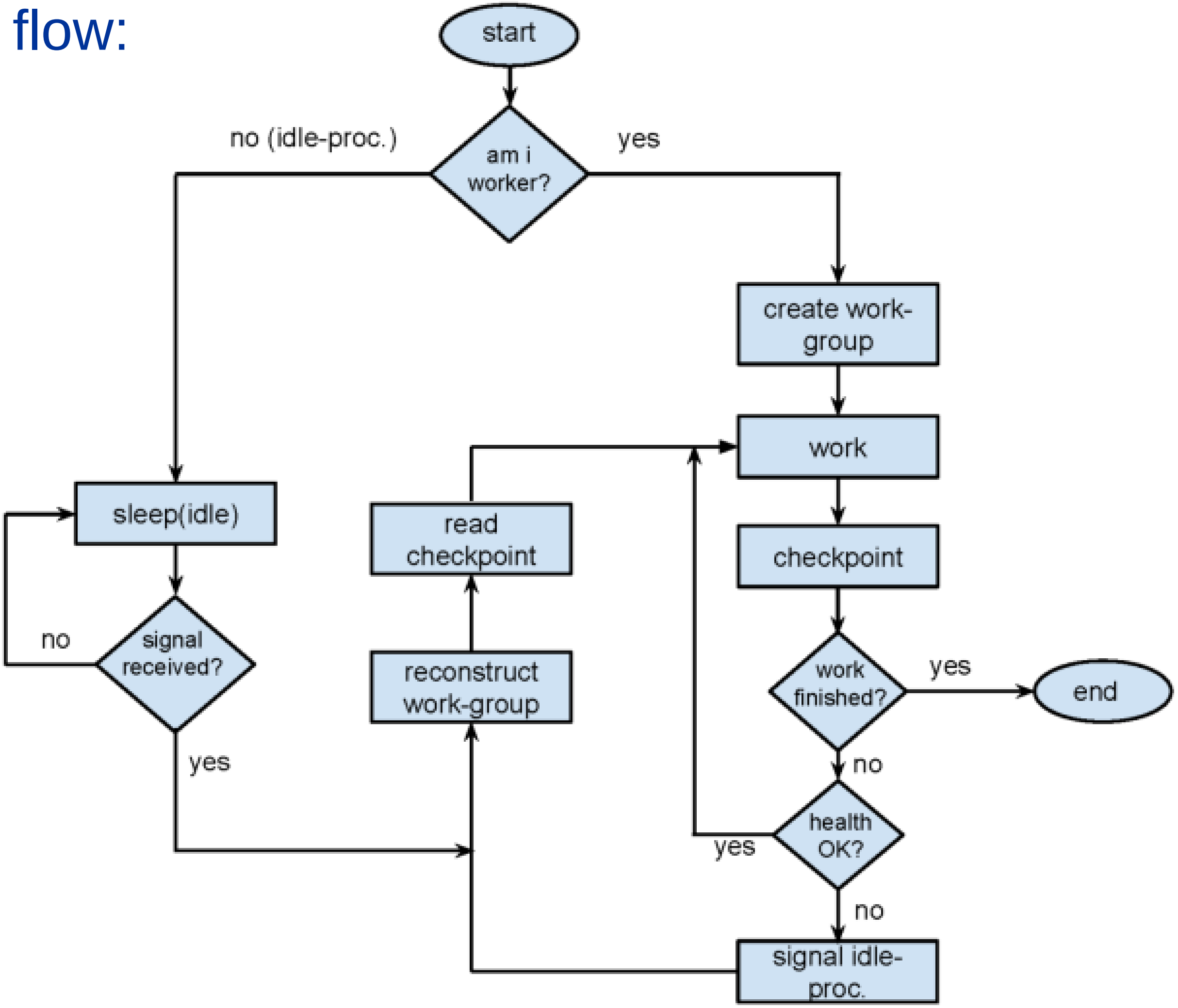
- **ABFT based application**

Toy FT implementation with LBM

- Idea:
  - Running the program with ‚n+m' processes, where ‚m' is the number of idle processes.
  - Program initially utilizes ‚n' processes for work (work-group)
  - In case of a failed process in ‚work-group', an idle process is added to the ‚work-group'.
  - Processes in newly established ‚work-group' restart the work from last checkpoint.

Toy FT implementation with LBM

- Program flow:

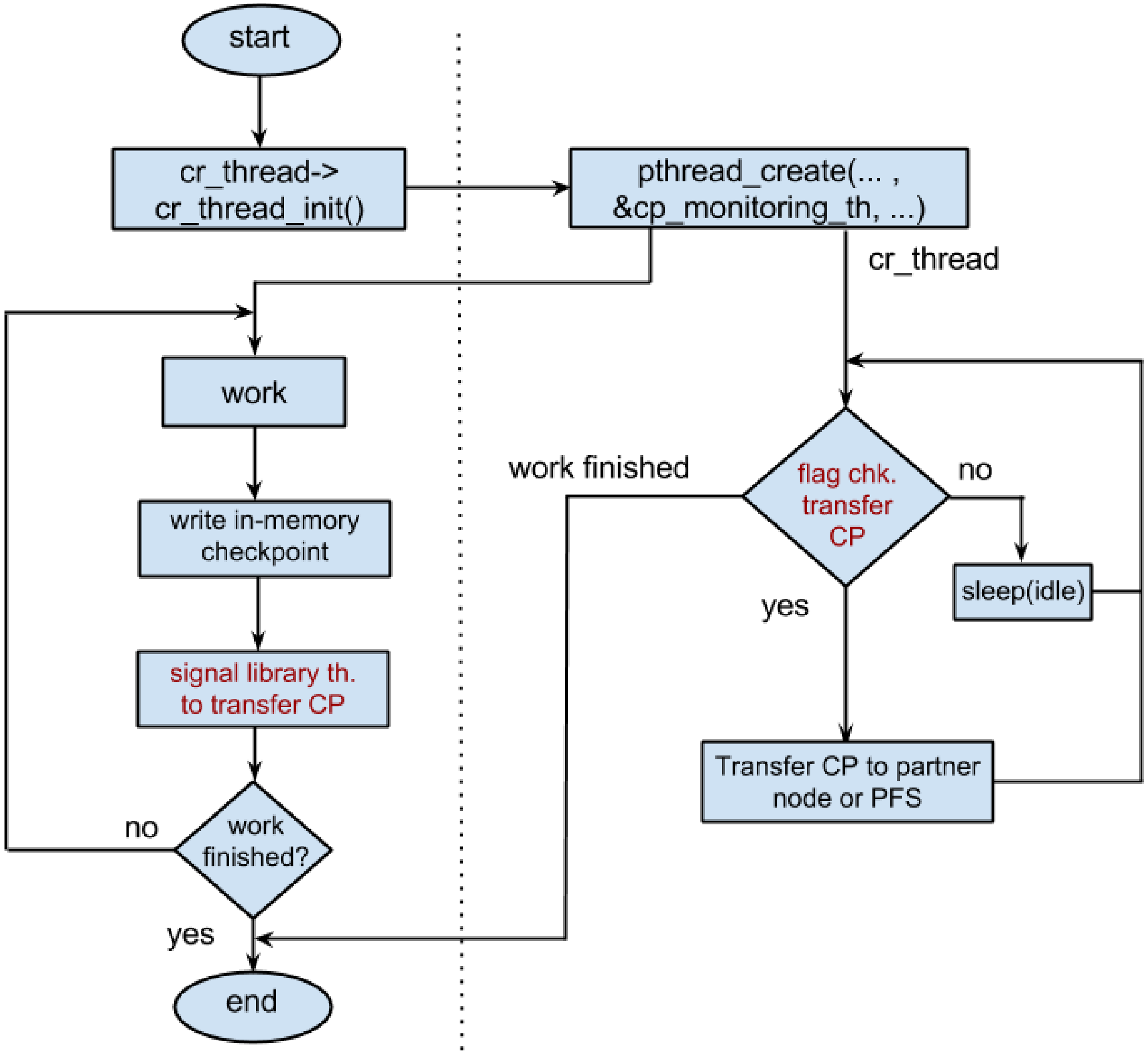


Neighbor level checkpointing for GPI (I)

- Devepment of Multi-level checkpointing infrastructure.
    - Based on library calls
    - Library thread responsible for transfering data in-between nodes and PFS.
    - Independent of communication library (MPI/GPI)

- Multi-level checkpointing with various layers of the application.
    - Different checkpoint frequency on various layers.

Neighbor level checkpointing for GPI (II)



Concluding remarks:

- Effective implementation of C/R and effective resource utilization can reduce overhead to minimum level.

- The overhead due to I/O bottlenecks can be reduced with asynchronous checkpointing approach.

- Node and neighbor-level checkpoints with occasional PFS-level checkpoints are highly scalable.

Thank you!

Questions?