Fault Tolerance and Checkpointing

- Sathish Vadhiyar
Introduction

☐ Checkpointing?

- storing application’s state in order to resume later
Motivation

- The largest parallel systems in the world have from 200,000 to 10 million parallel processing elements. (http://www.top500.org)
- Large-scale applications, that can use these large number of processes, are continuously being built.
- These applications are also long-running
- As the number of processing elements, increase the Mean Time Between Failure (MTBF) decreases
- So, how can long-running applications execute in this highly failure-prone environment? - checkpointing and fault-tolerance
Independent checkpointing

- Processors checkpoint periodically without coordination
- Can lead to domino effect - each rollback of a processor to a previous checkpoint forces another processor to rollback even further
Checkpointing Methods

1. Coordinated checkpointing
   1. All processes coordinate to take a consistent checkpoint (e.g. using a barrier)
   2. Will always lead to consistent checkpoints

2. Checkpointing with message logging
   1. Independent checkpoints are taken by processes and a process logs the messages it receives after the last checkpoint
   2. Thus recovery is by previous checkpoint and the logged messages.
Message Logging

- In message-logging protocols, each process stores
  - message contents and
  - sequence number

- To trim message logs, a process can also periodically checkpoint

- Once a process checkpoints, all messages sent/received before this checkpoint can be removed from log
Rules for Consistent Checkpointing

- In a parallel program, each process has events and local state
  - An event changes the local state of a process

- Global state - an external view of the parallel application (e.g. lines $S$, $S'$, $S''$) - used for checkpointing and restarting
  - Consists of local states and messages in transit

Figure 1. Global States.
Rules for Consistent Checkpointing

- Types of global states
  - Consistent global state - from where program can be restarted correctly
  - Inconsistent - Otherwise
Rules for Consistent Checkpointing

- Chandy & Lamport - 2 rules for consistent global states
  1. if a receive event is part of local state of a process, the corresponding send event must be part of the local state of the sender.
  2. if a send event is part of the local state of a process and the matching receive is not part of the local state of the receiver, then the message must be part of the state of the network.

- $S''$ violates rule 1. Hence cannot lead to consistent global state
Checkpointing Performance
Checkpointing Performance

- **Checkpoint overhead** - time added to the running time of the application due to checkpointing

- **Checkpoint latency hiding**
  - **Checkpoint buffering** - during checkpointing, copy data to local buffer, store buffer to disk in parallel with application progress
  - **Copy-on-write buffering** - only the modified pages are copied to a buffer. Other pages can be directly stored without copying to buffer. Can be implemented using `fork()` - forked checkpointing
Checkpointing Performance

- Reducing checkpoint size - memory exclusion and checkpoint compression
- Memory exclusion - no need to store dead and read-only variables
  - A dead variable is one whose current value will not be used by the program; The variable will not be accessed again by the program or it will be overwritten before it is read
  - Read only variable - whose value has not changed since the previous checkpoint
Incremental Checkpointing

- Memory exclusion can be made automatic by using incremental checkpointing
  - Store only that part of data that have been modified from the previous checkpoint
  - Following a checkpoint, all pages in memory are set to read-only
  - When the program attempts to write a page, an access violation occurs
  - During next checkpoint, only pages that have caused access violations are checkpointed
Checkpointing performance – using compression

- Using a standard compression algorithm
- This is beneficial only if the extra processing time for compression is lower than the savings that result from writing a smaller file to disk
- Redundancy/replication + checkpointing for fault tolerance
Replication

- Every node/process N has a shadow node/process N’, so that if one of them fail, the other can still continue the application - failure of the primary node no longer stalls the application.

- Redundancy scales: As more nodes are added to the system, the probability of failure of both a node and its shadow rapidly decreases.
  - Only one of the remaining n-1 nodes represent a shadow node.
Replication

- Less overhead for checkpointing
  - Higher checkpointing interval/period for periodic checkpointing
  - Recomputation and restart overheads are nearly eliminated

- Still need checkpointing: Why?
Total Redundancy
Partial Redundancy
Replication vs No Replication

[Graph showing the comparison between Replication and No Replication. The graph plots % Efficiency against Application-visible System Sockets. The green line represents Replication and the yellow line represents No Replication. The graph shows a significant decrease in % Efficiency for No Replication as the number of Application-visible System Sockets increases.]
References


References


- **MPICH-V2**: a Fault Tolerant MPI for Volatile Nodes based on the Pessimistic Sender Based Message Logging -- Aurélien Bouteiller, Franck Cappello, Thomas Hérault, Géraud Krawezik, Pierre Lemarinier, Frédéric Magniette -- To appear in *SuperComputing 2003*, Phoenix USA, November 2003
References

References for Replication

- Evaluating the viability of process replication reliability for exascale systems. SC 2011.
- Combining Partial Redundancy and Checkpointing for HPC. ICDCS 2012