Localized Fault Recovery for Nested Fork-Join Programs

Gokcen Kestor (Oak Ridge National Laboratory)
Sriram Krishnamoorthy (Pacific Northwest National Laboratory), Wenjing Ma (Chinese Academy of Sciences)

Introduction
- High performance computers are increasingly susceptible to errors
- Periodic checkpointing is widely used approach to fault tolerance, but recovery cost can be proportional to system size
  - It introduces large performance overhead
- We consider the design of fault tolerance mechanisms in the presence of fail-stop failures for
  - Nested fork-join programs,
  - Executed on distributed memory machines,
  - Load balancing provided by work stealing

Problem Statement and Objectives
- Reducing the amount of re-executed work in the presence of failures
- Guaranteeing forward progress even during fault recovery
- Ensuring correct interleaving of remote operations and error notifications
- Efficiently handling nested recovery, concurrent recovery, and failure-during-recovery scenarios

Our Proposal: ForkJoinFT
- A modified distributed-memory algorithm that incorporates efficient fault recovery
- ForkJoinFT executes all and only lost work due to a fault, it needs to:
  1. Track the relationship between the subcomputations performed by different threads
  2. Reconstruct the relationship among live processes that have
     pending interactions with the failed node
  3. Re-execute all and only lost subcomputations without interfering with
     the normal task execution

Tracking Global Computation
- We extended steal tree algorithm [PLDI’13] to retain only the live portion of
  subcomputations:
  - Each steal operation is identified with a unique ID (victim rank, working phase, level, and step)
  - At every steal operation, the thief gets its victim steal path and adds the
    current steal operation
  - All the preceded steals (Stolen Step)
    from a given victim in the same working phase are recorded

Recovering Global Computation
- Failure notifications are assumed to be sent to the server threads
- Upon a failure notification, each server thread independently initiates recovery:
  - Identifies pending subcomputations stolen by the failed worker
  - Marks the victim of the failed worker as a recovery node
  - Requests steal tree paths that include the failed worker from all workers
  - Collects all steal tree paths and constructs a replay tree
  - The root of the replay tree is the subcomputation stolen by the failed worker
  - Collection is a distributed binary-tree-based reduction
  - Makes the replay tree and its root task ready to be stolen

Scheduling Re-Execution
- When a thief steals work to be re-executed:
  - Its victim determines the task’s frontier
  - Task’s frontier is the failed worker’s list of alive children
  - The thief assumes ownership of the root task of replay tree
  - Thieves of this subcomputation will return their results to new owner, rather than the failed worker

Results
- Negligible overhead and does not increase with core count
- Space overhead per thread is generally a few KB and remains roughly constant when scaling to larger node counts

Conclusions
- We presented an approach to localized fault recovery specific to
  nested fork-join programs executed on distributed-memory systems
- Our fault-tolerance approach:
  - Introduces negligible overhead in the absence of faults, within
    the execution time variation
  - Re-executes all and only lost work due to faults
  - Significantly decreases the amount of work re-executed as compared to alternative strategies
  - Presents a recovery overhead roughly proportional to the amount of lost work