CAF 2.0 Phasers

Sri Raj Paul

Advisor: John Mellor-Crummey
Department of Computer Science
Rice University
sriraj@rice.edu
Scale of Computation

- Data Movement
- Synchronization
- Load Balancing
- Power Consumption

http://www.scl.ameslab.gov/Projects/parallel_computing/cluster_examples.html

http://www.olcf.ornl.gov/titan/

CHAOS

http://nepaliaustralian.com/2012/09/07/traffic-chaos-around-the-world/
Patterns of Synchronization

• Point-to-Point

• Producer-Consumer

• Barrier
Phaser

- Unifies barrier and point-to-point synchronization
- Allows dynamic addition/deletion of processes
- Flexible participation modes for each process involved
Phaser Constructs

• Registration
  — phaser_create(...) : creates a phaser in a specified mode
    – SIGNAL
    – WAIT
    – SIGNAL_WAIT
  — spawn(...) : adds a new process to an existing phaser

• Synchronize
  — next(...) : synchronizes according to the mode
Producer-Consumer Example

if (ME == 1)
    phaser_create(ME, ph, SIG, ...)
    spawn<WAIT, ph>(fn, ...)[2]
    produce_data()
    next(ph)
.
.
.

fn(...)  
    next(ph)
    consume_data()
.
.
.

end

Producer: Process 1

Consumer: Process 2

Process 1

Process 2
Design Challenges

• Scalability
  — Need to support thousands if not millions of participants

• Concurrency
  — Scalable parallelism requires concurrent operations

• Distribution
  — All significant operations involve interactions between multiple agents

• Dynamism
  — Must support dynamic addition/deletion of processes

• Correctness
  — Operations must be free of deadlock and livelock
Skip Lists as a Building Block for Phasers

- Probabilistic replacement to balanced trees
- Addition/deletion without rebalancing
- Logarithmic space/time complexity for operations

https://github.com/tewuapple/SkipList
Propagate Signals Using Skip List

- One skip list for signalers and one for waiters
- Signaler root collects the signals from all signalers
- Passes it to waiter root
- Waiter root distributes signal to all the waiters
Operations to Maintain the Skip List

• Creation
  — Recursive doubling

• Addition
  — Spawnee included into the skip-list before the spawn call returns
    – eager-single-link-modify (to avoid blocking of spawner)
    – lazy-multi-link-modify (move to the required height)

• Deletion
  — Lazy level by level deletion
Single Level Addition to a Skip List

Find current signal to neighbor
Find next signal to neighbor
Find right neighbor at level
Ready for move up
Acknowledgements
Verification of the protocol

• Phaser protocol involves many participants
  — Addition/deletion-signaling happens simultaneously
  — Too many messages in flight
  — Proving properties is non-trivial

• Model Checking is the solution
  — Explore the whole state space, i.e., all potential interleaving

• Challenges
  — Size of the state space increases exponentially with the number of participants
  — Exhaustive search not possible
    – approximate methods are necessary
Verification using SPIN

• SPIN
  — Tool to automate verification of large distributed systems
  — Write an algorithm to be checked in PROMELA
  — Approximate Model-checking capability
  — Progress and correctness properties expressed in Linear Temporal Logic

• Phaser properties currently modeled
  — No signals are lost
  — Eventually neighbors should become consistent

• Our approach
  — Analyze sufficient set of interleavings to drive an agent through all configurations of the phaser protocol
Summary

• Phaser unifies barrier and point-to-point synchronization

• Skip list used as backbone structure
  — Scalability
  — Flexibility

• Protocol Verification done using SPIN
  — Employ approximate methods to model check the entire state space