

## Parallel and Distributed Systems

Instructor: Sandhya Dwarkadas  
Department of Computer Science  
University of Rochester

## Distributed Systems Issues

- Concurrent access
- Scalability
- Availability
- Reliability and failure transparency

## Global State Determination

- Example uses: deadlock or termination detection
- Solution: distributed snapshot (Chandy and Lamport 1985)
  - Important property: consistent global state/cut
  - Termination detection: snapshot in which all channels are empty

## Faults and Failures

- Fault – cause of an error that might lead to failure
  - Transient
  - Intermittent
  - Permanent
- Failure models
  - Crash (fail-stop)
  - Omission
  - Timing
  - Response (value or state is incorrect)
  - Byzantine or arbitrary

## Fault Tolerance

- Availability – most likely working at any instant
- Reliability – run continuously without failure (increase time interval between failures)
- Safety – nothing catastrophic happened on temporary failures
- Maintainability – ease of repair

## Agreement in Faulty Systems

- A hard problem
  - Perfect processes, unreliable communication
    - E.g., the two-army problem
  - Unreliable processes, perfect communication
    - E.g., the Byzantine generals' problem

## Logical Clocks

Agreement on ordering of events rather than what time it is is what matters

- Lamport Timestamps: partial order
  - “happened-before” relation “ $\rightarrow$ ”, causal ordering, or potential causal ordering
    - Transitive relation
  - Assign every event a time value  $C(a)$  such that if  $a \rightarrow b$  then  $C(a) < C(b)$  – can be captured numerically through a monotonically increasing software counter
  - Lamport's solution for message ordering
    - Each message carries sending time
    - Receiver's clock is set to the greater of its own clock or the sender's clock and then incremented by 1
    - Between every two events, the clock must tick at least once

## Vector Timestamps

- Lamport timestamps do not imply causality
- Solution: vector timestamps: captures the notion of causality in addition to concurrency
  - Maintain a vector clock with  $n$  integers for  $n$  processes

## Transactions

- Modify multiple data items potentially at multiple locations/by multiple processes as a single atomic operation
- Transaction properties (ACID) –
  - Atomic – happens indivisibly to the outside world
  - Consistent – does not violate system invariants – must hold before and after but not necessarily during
  - Isolated (or serializable) – refers to multiple simultaneous transactions – the final result must appear as if each transaction occurred in some sequential order
  - Durable – once committed, the results become permanent – no failure can undo the results

## Distributed Systems Implementation

- Logical versus physical clocks
  - Lamport and vector timestamps
- Distributed snapshots
  - Consistent global state
- Coordinator election
- Mutual exclusion

## Distributed Transactions

- ACID properties
- Concurrency control
  - Two-phase locking
  - Pessimistic vs. optimistic timestamp ordering
- Distributed commit
  - Two-phase vs. three-phase commit

## Group Communication

- Virtual synchrony – message multicast to group view  $G$  is reliably delivered to each non-faulty process in  $G$  or to none.
- Unordered vs. FIFO-ordered vs. causally-ordered vs. totally-ordered message delivery
- Scalable reliable multicast

## Fault Tolerance

- Checkpointing with rollback recovery
  - Independent checkpointing
  - Coordinated checkpointing
    - Two-phase blocking vs. distributed snapshots (non-blocking)
  - Communication-induced checkpointing
- Log-based recovery
  - Execution of a failed process during recovery is identical to its pre-failure execution

## Programming Models

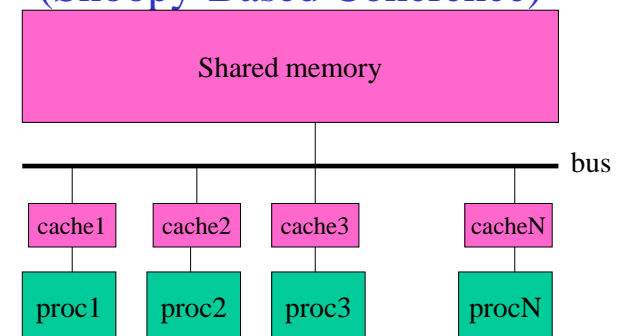
- Standard models of parallelism
  - shared memory (Pthreads)
  - message passing (MPI)
  - data parallelism (Fortran 90 and HPF)
  - shared memory + data parallelism (OpenMP)

## Parallel Processing Issues

- Data sharing
- Process coordination
- Distributed (NUMA) vs. centralized (UMA) memory
- Connectivity
- Fault tolerance

Correctness and performance issues: deadlock, livelock, starvation

## Shared Memory Hardware (Snoopy-Based Coherence)





## Basic Hardware Mechanisms for Synchronization

- Test-and-set – atomic exchange
- Fetch-and-increment – returns value and atomically increments it
- Load-locked/store conditional – pair of instructions – deduce atomicity if second instruction returns correct value

## Software Synchronization Algorithms

- Locks - test&test&set, ticket, array-based queue, MCS (linked list)
- Barriers – centralized/sense-reversing, software combining trees, tournament, dissemination
- Lock-free, non-blocking, wait-free properties

## Sequential Consistency

- "A system is sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program." [Lamport 79]
  - In practice, this means that every write must be seen on all processors before any succeeding read or write can be issued

## Consistency Model Classification

- Models vary along the following dimensions
  - Program order (for each processor)
  - Write atomicity

## Course Recap

- Parallel programming models
- Process coordination – synchronization and data coherence and consistency
- Scalability and group communication
- Fault tolerance and reliability

## Course Recap

- Basics of parallelization: dependences, synchronization, patterns of parallelism
- Parallel languages/models: PGAS, GPGPU, MPI, OpenMP, Cilk, pthreads
- Coherence, synchronization, and consistency in hardware and software
- Transactional memory
- Fault tolerance and recovery
- Scalability and group communication