Maekawa's Algorithm

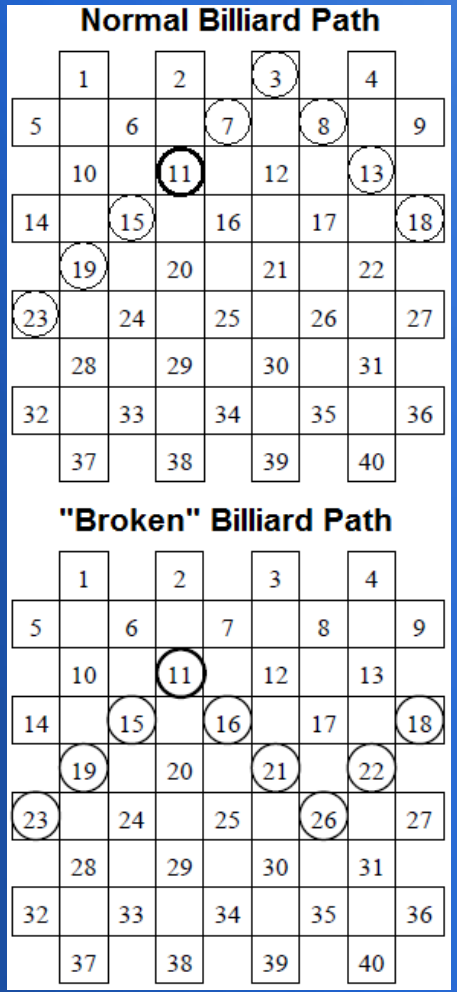
Jordan Adamek

Maekawa's DME Algorithm

- Permission-based distributed mutual exclusion
- Communication through Channels
- 6 Major Messages:
  - 3 for main functionality: Request, Grant, Release
  - 3 for deadlock avoidance: Inquire, Failed, Yield
- Requests are time-stamped
- Quorums

Billiard Quorums

- Each quorum is of size root(2 * N + 1)
  - e.g: for 40 processes, quorum size = 9
- Quorum constructed from so-called “Broken Billiard Path”
  - Normal billiard path is the diagonal billiard shot from x or y axis to target process in a modified (odd) grid
  - Broken path ensures quorum uniqueness for processes on an otherwise equivalent path

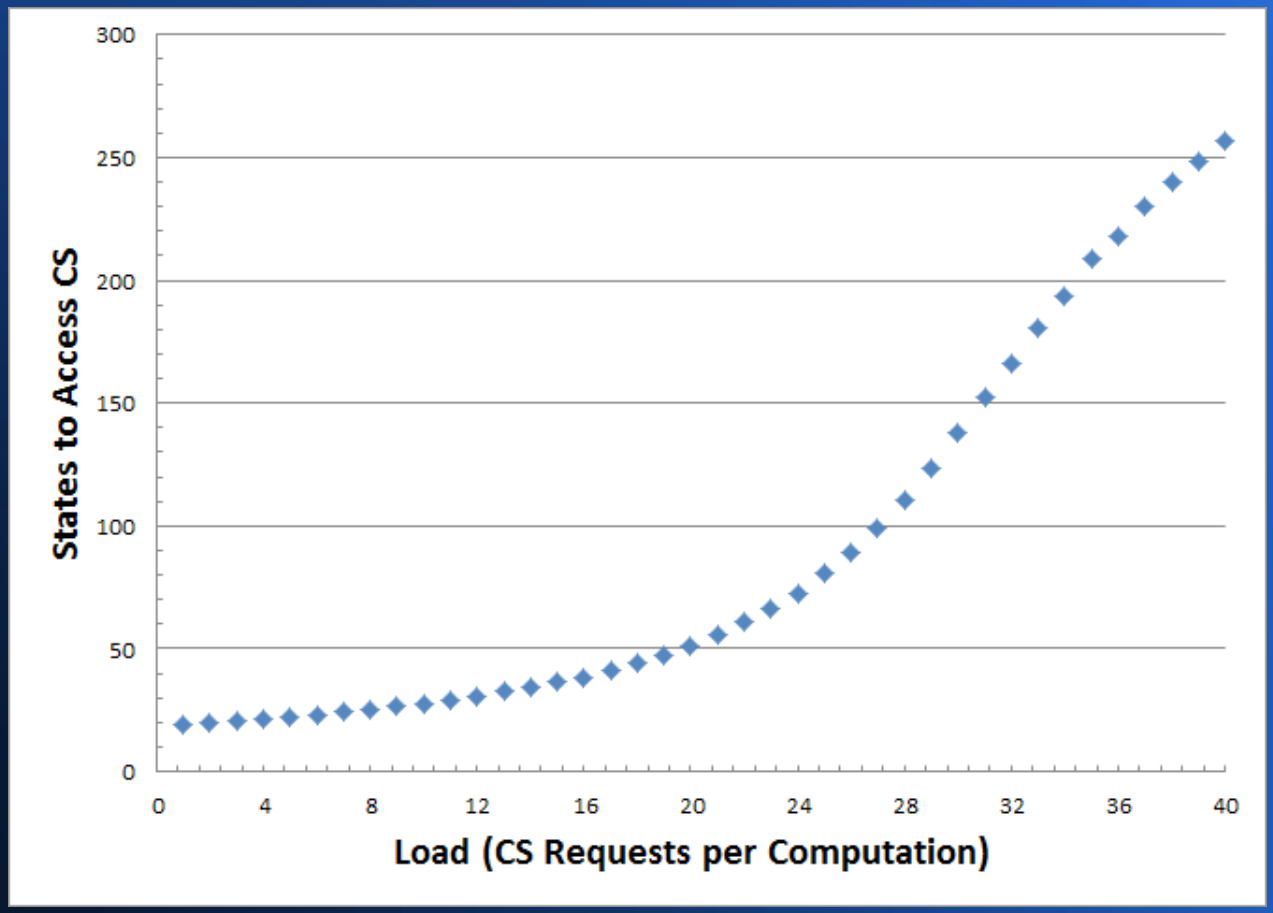


Experimental Procedure

- Interleaving Semantics
- 40 Processes
- Billiard Quorums of size 5
- 1,000 states per computation
- Experiments varied system load from 1 to 40 (system size):
  - CS requests scheduled in an array to be sent out over the course of the computation
  - Number of total requests sent determined by load
- 10,000 computations per data point (load value)
- Measured two metrics: latency and through-put (efficiency)

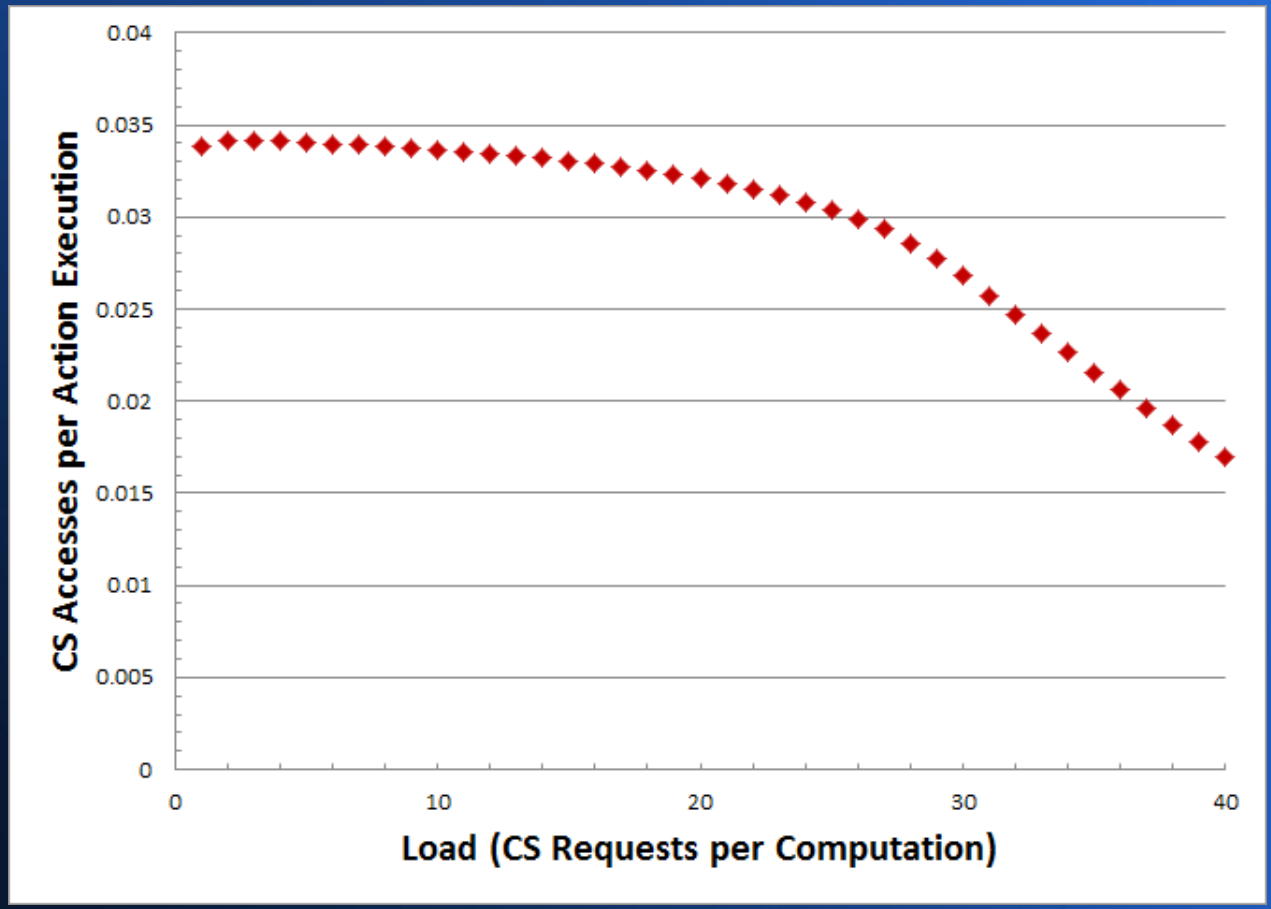
Latency

States from CS Request to CS Access



Throughput

CS Accesses per Action Executed



Future Work

- Vary any of the following:
  - Quorum size and construction method (original Maekawa Quorums vs. Billiard Quorums)
  - System size
  - Execution semantics (Synchronous, Power Set)
- Search for alternative/less costly methods to implement deadlock avoidance

Simulation Structure

- Four main classes comprise the engine:
  - Action: Guarded action using a boolean *guard* method and a void *command* method. Contains a pointer to the process which defines it; both methods act only on this pointer.
  - Process: Distributed process which contains all local variables and assigned actions defined by the target algorithm.
  - Network: Comprises the collection of processes and defines any initial relationships between them, such as neighbors in a graph or quorum members.
  - Engine: Contains a pointer to a network object and runs computations for a given number of maximum states (or until deadlock). Each state, checks for enabled processes and, according to the engine's execution semantics defined by an enumerable member variable, calls for processes to execute a random enabled action.

Simulation Structure (cont.)

- Main classes extended for implementation of Maekawa's algrorithm.
- Quorum and Channel classes defined.
- Each action in Maekawa's extended from the base Action class, definining the pure virtual methods *guard* and *command*.
- Maps (associative arrays) used to relate quorum members to associated variables (such as waiting for permission, <failed> messages received, etc.).

Class Co-dependence

- Problem: we wish to define two classes, A and B, which should contain an object whose class is of the other.
- In sequential C++ programming, a class must be already declared for an object to be declared with that type.
- Class A must be defined before class B for B to have member variable of class A, but likewise must be done for class A to contain a variable of class B; one or the other must be defined first.

Class Co-dependence

Solution:

- Empty declare one of the classes, e.g class A
- Define B with a pointer to an object of class A
- Define class A with any number of objects of class B
- Both implementations must follow all three declarations

Use of Co-dependence in Distributed Algorithm Simulation

- Action class contains a pointer to a Process, and may freely reference the exact process that defines in its *guard* and *command* methods without any iteration over the network or other means.
- Processes contain arrays of Action pointers. May iterate over these actions to check for enabled ones, and execute their commands without having to know the exact nature (or name) of the action. Any number or complexity of actions may be defined to a process.
- Engine is able to simulate computations of guarded command algorithms regardless of what algorithm is being simulated