EVALUATION
OF
CONCURRENT POOLS

DAVID KOTZ

CARLA ELLIS

DUKE UNIVERSITY
DURHAM, NC

What is a pool?

A pool is a data structure representing a collection of objects, tasks, resources, etc.

We can add items to the pool, and remove an <u>unspecified</u> item from the pool.

What is it used for?

When it doesn't matter which item is removed from the pool, e.g., processing a set of tasks in any order, possibly creating new tasks simultaneously.

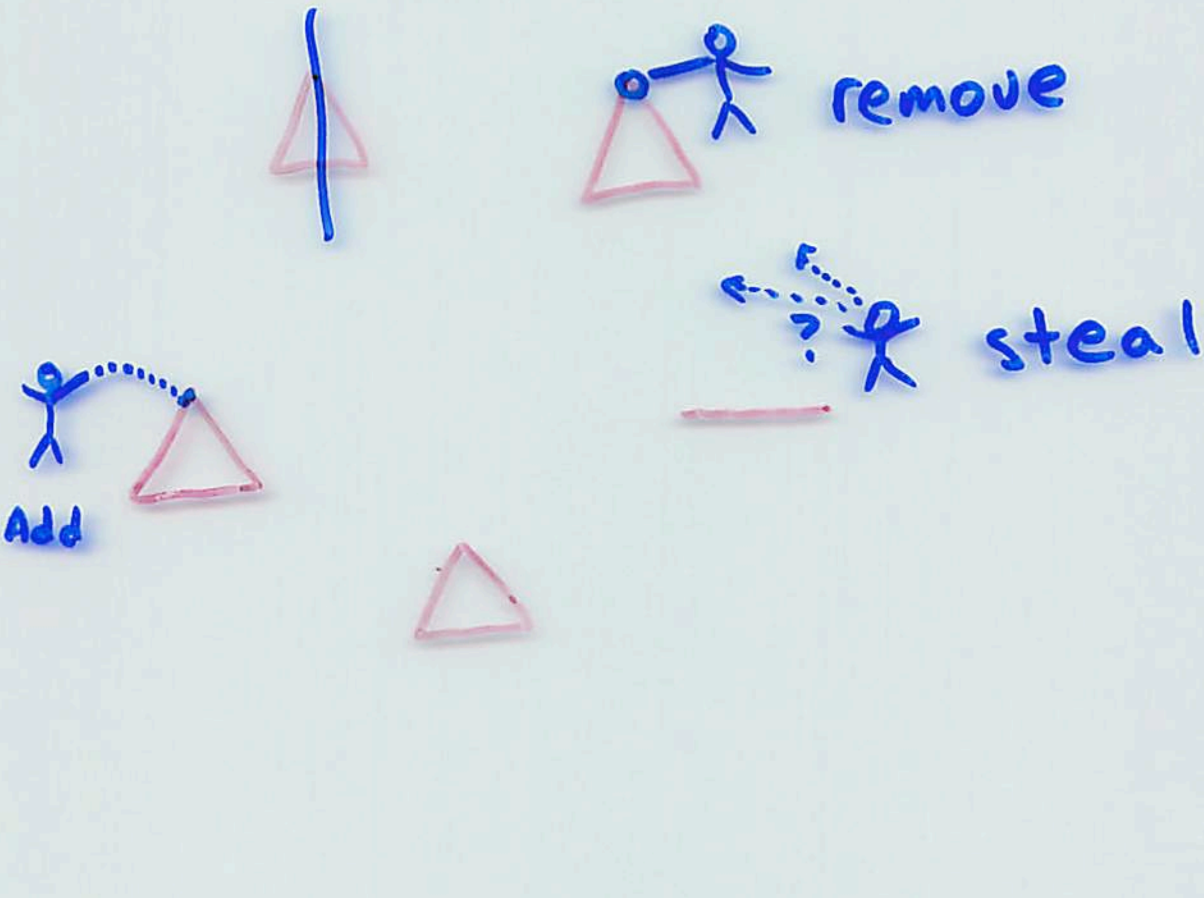
A concurrent pool supports simultaneous access by many processes.

CONCURRENT POOL

Goals:

- avoid interference between procs
- maintain locality of reference
- efficient access for all processes

Partition the pool into local segments, one per process:



OUR STUDY

Compares 3 implementations of concurrent pools.

The **steal** operation becomes the dominant factor in performance:

- interference with others
- non-local references

We compare 3 search algorithms

- TREE
- LINEAR
- RANDOM

TREE SEARCH

[Manber, SIAM J. Comput. 11/86]

Superimpose a tree on the segments:

internal nodes have a marker for each child that indicates whether the child's subtree is <u>empty</u>

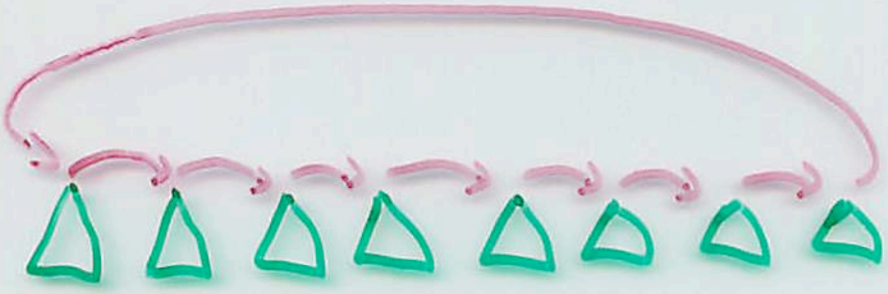
processes traverse the tree looking for non-empty segments

subtrees are marked empty as processes find them to have all empty segments

begin the search where elements were last found

LINEAR SEARCH

Try the segments in some fixed order



no information is kept about empty segments

begin the search where elements were last found

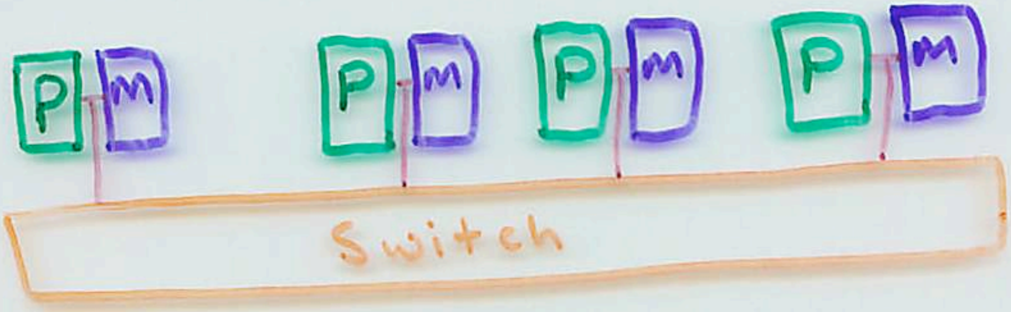
RANDOM SEARCH

Try the segments at random:

EXPERIMENTS

16-process pools on a 32-node BBN Butterfly I

NUMA architecture (4:1)



(Note similarity to LAN)

Implementations use local memory for add and remove, but must use remote memory for steals.

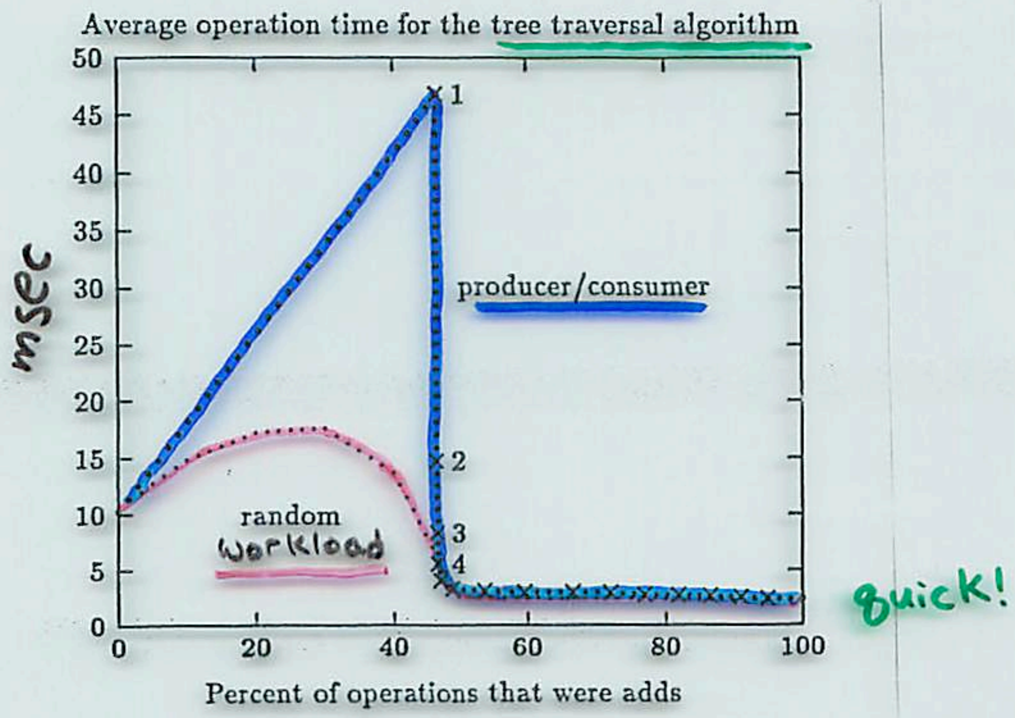
WORKLOAD

- STRESSFUL
- ENCOURAGE STEALS

TWO TYPES:

- RANDOM
  - All processes add and remove
  - varying mix of add / remove
    - Sparse: < 50% adds
    - sufficient: ≥ 50% adds
- PRODUCER / CONSUMER
  - Some processes add
  - Some processes remove
  - Vary number of producers

RESULTS



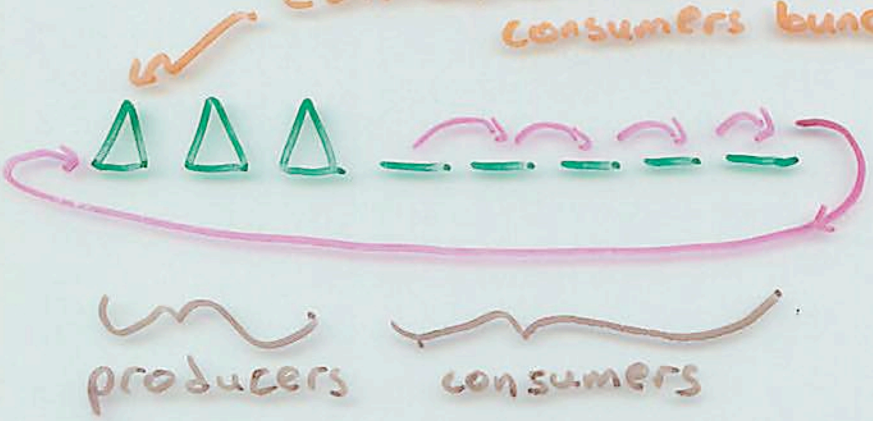
Sparse ← | → sufficient

(steals dominate) (steals are rare)

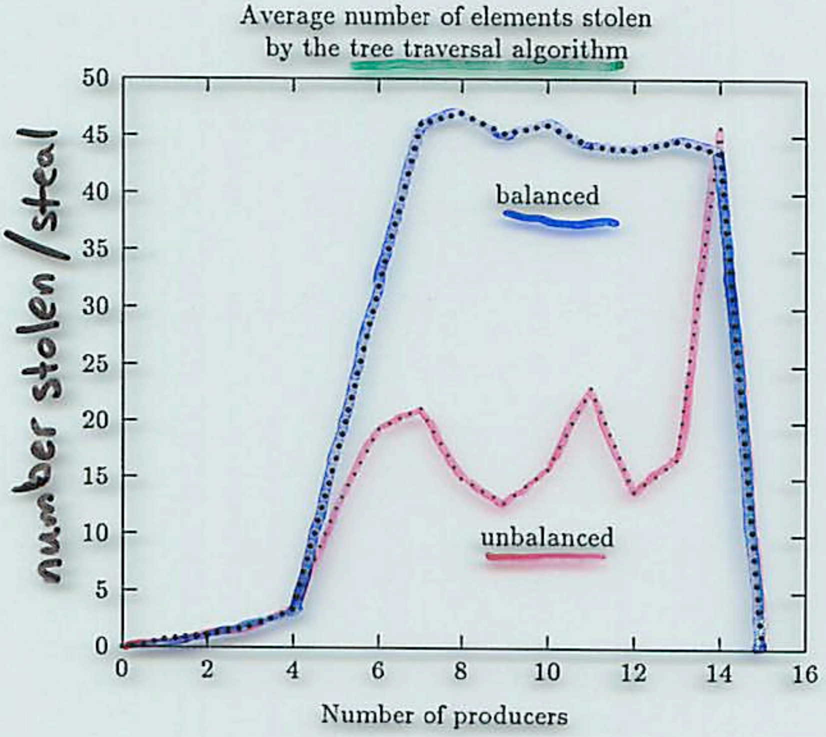
As expected, steals determine performance

BUNCHING

contention here for stealing
consumers bunch up



consumers find different producers



balancing the producers allows consumers to steal more on each steal, and to interfere less.

COMPARING ALGORITHMS

- All 3 similar for producer/consumer
- All 3 similar for random, sufficient loads
- Tree search slow for random, sparse loads

This despite the fact that the tree search

- examines fewer leaves
- steals more elements

→ lower overhead?

No, internal nodes of the tree add overhead as a source of contention.

Traversing tree nodes similar in time to examining a pool segment.

Tree search better for higher remote : local access time ratio?

e.g., distributed system

EXPERIMENT:

artificially delay access to "remote" parts of the pool (leaves, tree nodes)

Delays from 1 μsec ... 100 msec

(Normal operation time ~ 100 μsec)

RESULT:

- Tree search never wins
- All 3 become similar with high delay

Conclusion:

Complexity of tree search not worth it.

APPLICATIONS

3-D Tic-Tac-Toe game

Minimax search tree of 3 levels
and 250,000 nodes.

Algorithm:

Add nodes to central work list.
Remove node from list.
Process node.
Generate new nodes to put on list.

Concurrent Pool:

Pool for work list.

Speedup of 14.6-15.4 on 16 procs.
(Three search algorithms similar.)

Global stack:

used a simple, single stack for work list.

Algorithm identical.

40% slower than pool

speedup of only 10.7 on 16 procs.

SUMMARY

Concurrent pools provide an efficient, highly concurrent solution to many problems.

Locality of reference good - steal only when necessary - good for NUMA architectures.

Complex tree search not worth the effort needed to maintain the high locality, low interference.

Considerations of overhead need to be taken into account in design of concurrent data structures, including locality in NUMA architectures.

FUTURE WORK

- other implementations of tree search algorithm?
- Larger pools? (>16 processes)
- Other search algorithms?

A talk by David Kotz, June 1989.

Presented at the Ninth International Conference on Distributed Computer Systems (ICDCS), in Newport Beach, California.

Describing the paper by David Kotz and Carla Ellis, "Evaluation of concurrent pools." In *Proceedings of the Ninth International Conference on Distributed Computer Systems (ICDCS),* pages 378-385. IEEE Computer Society Press, 1989.