A Universal Construction to implement Concurrent Data Structure for NUMA-multicore

Zhengming Yi

National University of Defense Technology

Changsha, China

yizhengming17@nudt.edu.cn
Concurrent Data Structures (CDS)

Used everywhere: kernel, libraries, applications

Issues:
- High copying overheads
- NUMA-oblivious design
- Read-side overhead
- Complex
Goals

• Design a new Universal Construction (called CR), which transforms a sequential implementation of a data structure into a concurrent implementation

➢ Provide efficient read-side performance
➢ Provide scalable write-side performance on NUMA-multicore
Our method: CR

- CR relies $n+1$ replicas of the data structure where $n$ is the number of NUMA nodes, reader-writer lock, delegation and a shared log.
Our method: CR

CR relies n+1 replicas of the data structure where n is the number of NUMA nodes, reader-writer lock, delegation and a shared log.

- Efficient Read: Keep one up-to-date replica for read-only access at all times
Our method: CR

CR relies $n+1$ replicas of the data structure where $n$ is the number of NUMA nodes, reader-writer lock, delegation and a shared log.

- Efficient Read: Keep one up-to-date replica for read-only access at all times
- NUMA-aware write: Use a shared log to synchronize cross-nodes threads and use delegation to synchronize local threads
Our method: CR

CR relies n+1 replicas of the data structure where n is the number of NUMA nodes, reader-writer lock, delegation and a shared log.

- Efficient Read: Keep one up-to-date replica for read-only access at all times
- NUMA-aware write: Use a shared log to synchronize cross-nodes threads and use delegation to synchronize local threads
- Without requiring inner knowledge of the data structure
Structure Chart for CR

1. Delegation within a node
2. Shared log across nodes
3. CurReplica for read-only
Delegation: Collect requests within a NUMA node 1

NUMA Node 1
- Readers
  - Writer 1
    - Req1
  - Writer 2
    - Req2
  - Writer 3
    - Req3
  - Writer 4
    - Req4

NUMA Node 2
- Readers
  - Writer 5
    - Req5
  - Writer 6
    - Req6
  - Writer 7
    - Req7

Shared log
- Replica 1
  - Gtail
- Replica 2
- Replica 3
Delegation: Write requests within a NUMA node 1

NUMA Node 1
- readers
  - writer 1 (req1)
  - writer 2 (req2)
  - writer 3 (req3)
- writer 4 (req4)
  - server

NUMA Node 2
- readers
  - writer 5 (req5)
  - writer 6 (req6)
- writer 7 (req7)
  - server

Shared log
- req1
- req2
- req3
- req4

Replica 1
- ltail

Replica 2

Replica 3
- ltail

gtail
Shared log: Execution for requests from a NUMA node 1
Transition CurReplica to Replica 1
Transition CurReplica to Replica 1
Delegation: Collect and write requests within a NUMA node 2

NUMA Node 1
- writers (req1, req2, req3)
  - writer 1
  - writer 2
  - writer 3
- reader
  - req4

NUMA Node 2
- writers (req5, req6)
  - writer 5
  - writer 6
- reader
  - req7

CurReplica
- req1
- req2
- req3
- req4
- req5
- req6
- req7

Shared log
- gtail

Replica 1
- req1
- req2
- req3
- req4
- req5
- req6
- req7

Replica 2

Replica 3
- req7

Server
Shared log: Synchronization and execution for requests
Transition CurReplica to Replica 3

NUMA Node 1
- readers
  - writer 1
  - writer 2
  - writer 3
- server
  - writer 4

NUMA Node 2
- readers
  - writer 5
  - writer 6
- server
  - writer 7

CurReplica
- req1
- req2
- req3
- req4
- req5
- req6
- req7

Shared log

Replica 1
- Itail

Replica 2

Replica 3
- Itail

gtail
Transition CurReplica to Replica 3

- Replica 2
- Replica 3
- Shared log
- NUMA Node 1
  - readers
  - writer 1
    - req1
  - writer 2
    - req2
  - writer 3
    - req3
  - writer 4
    - req4
- server
- NUMA Node 2
  - readers
  - writer 5
    - req5
  - writer 6
    - req6
  - writer 7
    - req7
- server
Results

Sever:
2 NUMA nodes
14 cores/node + hyperthreading
(total 56 hardware threads)
Skiplist Priority Queue – 10% Updates

The graph shows the performance of different data structures under varying thread counts and configurations.

- **PSim**: Blue crosses
- **PSimOpt**: Red squares
- **CR**: Purple triangles
- **CX**: Green circles
- **CXTimed**: Blue stars
- **LF**: Red diamonds
- **SL**: Brown dots

The graph contrasts performance in a single node scenario versus greater than one NUMA node scenarios. The vertical arrows indicate performance improvements:
- **2.1X** improvement in single node configuration
- **5.4X** improvement in NUMA node configuration

The horizontal axis represents the number of threads, with a focus on 10% update scenarios.
Using CR in KyotoCabinet – 2% Updates

![Graph showing performance comparison between CR, CXTimed, CX, and vanilla with 2% updates. The graph indicates a performance gain of 18.1X.](image)
Conclusion: CR Works Well

• Keep one update-to-date replica for read-only access at all times
• Use a shared log to synchronize cross-nodes threads
• use delegation to synchronize local threads
Thank you!