Black-box Concurrent Data Structures for NUMA Architectures

Irina Calciu (VRG)
Siddhartha Sen (MSR)
Mahesh Balakrishnan (Yale)
Marcos K. Aguilera (VRG)
Concurrent Data Structures (CDS)

Used everywhere: kernel, libraries, applications

Issues:

• Difficult to design and implement
• Complex and rigid
• Assume uniform memory
Non-Uniform Memory Access (NUMA)
Goals

• Design efficient NUMA-aware CDS

• Black-box method: works for any data structure

• Application level
Transform sequential DS to NUMA-aware CDS

e.g. insert(A), remove(B), lookup(C)

Sequential data structure

NUMA-aware concurrent data structure

App Thread 1

op (e.g. “insert(X)”)  
resp (e.g. “OK”)

NUMA Node 1

App Thread 1

resp

App Thread 2

resp

App Thread 3

resp

App Thread 4

resp
API

\[ S.\text{ExecuteSeq}(\text{op, args}) \rightarrow \text{Result} \]

\[ N.\text{Execute}(\text{op, args}) \rightarrow \text{Result} \]

\[ N.\text{IsReadOnly}(\text{op}) \rightarrow \text{Boolean} \]
Our Method: Node Replication (NR)

NUMA Node 1

Local Replica

Synchronization within a node

Synchronization between nodes

Synchronization within a node

NUMA Node 2

Local Replica

Sequential data structure
Synchronizing Replicas (Cross-node)

NUMA Node 1
Local Replica

NUMA Node 2
Local Replica

Shared Log
Synchronizing Replicas (Cross-node)

NUMA Node 1

NUMA Node 2

Shared Log

- remove Z
- insert A
- insert B
- insert C
- insert D
- insert E
- insert N

Local Replica

Local Tail

LogTail

Log grows down
Synchronizing Each Replica (Intra-node)

**NUMA Node 1**
- Local Replica

Access to replica?

**Flat Combining**

**NUMA Node 2**
- Local Replica

Access to replica?

**Flat Combining**

**Shared Log**
Access to Replica

NUMA Node

Local Replica

Local Tail

Thread
Update req

Thread
Read req

Thread
Update req

Thread
Update req

Flat Combining: [Hendler et al., 2010]
Access to Replica

NUMA Node

Local Replica

Local Tail

Thread
Update req

Thread
Read req

Thread
Update req

Thread
Read req

Flat Combining: [Hendler et al., 2010]
Putting It All Together

NUMA Node

Local Replica

Local Tail

Thread

Thread

Thread

Thread

Shared Log

LogTail

empty

empty

empty
Algorithm Summary: Replication, Log, Combining

NUMA Node 1
- Local Replica
- Access to replica?
- Flat Combining

Shared Log

NUMA Node 2
- Local Replica
- Access to replica?
- Flat Combining
Server:
4 NUMA nodes
14 cores/node + hyperthreading
(total 112 hardware threads)
Skiplist Priority Queue – 10% Updates

(NR) Node Replication
(FC+) FC + RWL
(RWL) Readers-Writer Lock
(LF) Lock-free
(FC) Flat Combining
(SL) Spinlock

ops/us

# threads

1 28 56 84 110

1.7X

6X
Using Replication in REDIS: 10% Updates

- (NR) Node Replication
- (FC+) FC + RWL
- (RWL) Readers-Writer Lock
- (FC) Flat Combining
- (SL) Spinlock

Graph showing the performance of replication methods with 10% updates. The y-axis represents operations per second (ops/us) and the x-axis represents the number of threads. The graph shows different performance levels for each method, with a 2.6X improvement noted.
Rationale

- Trade memory + computation for less communication
  - Compact representation of operations
  - Limited cross-node synchronization and contention
- Enable parallelism
  - Combiners across nodes
  - Readers within a node
  - Readers and the combiner on the same node
- Leverage batching and reordering
Conclusion: NodeReplication Works Well

• Black-box: works for any data structure
• Good for small and medium size CDS
• Beneficial for contended CDS
Thank you!

icalciu@vmware.com

https://research.vmware.com/