A Lightweight Log-based Deferred Update for Linux Kernel Scalability

Joohyun Kyong and Sung-Soo Lim

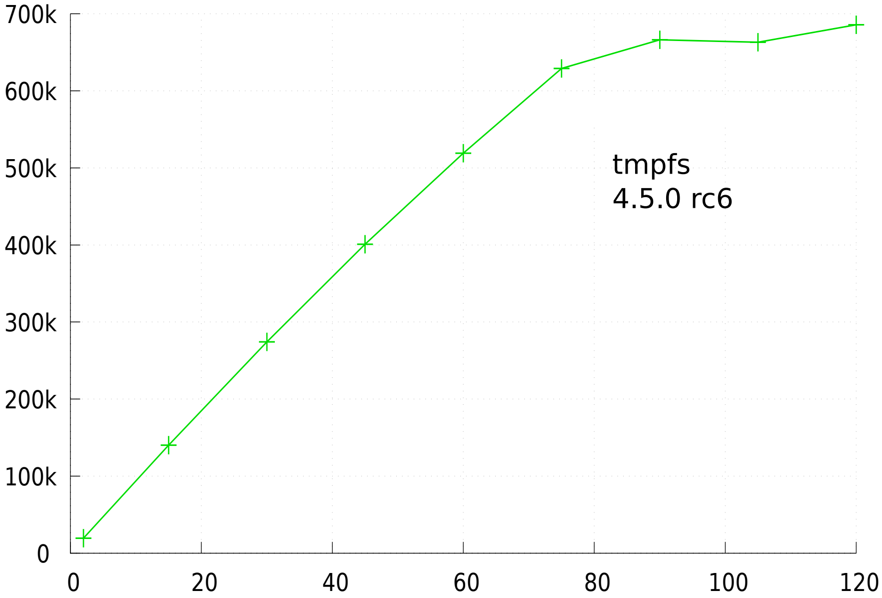
School of Computer Science
Kookmin University

August 26, 2016

Outline

- Background of research
- Our new method and Evaluation
- Future plans and Summary

AIM7 Scalability – Linux 4.5.0



Operating System Scalability

Scalable operating system.

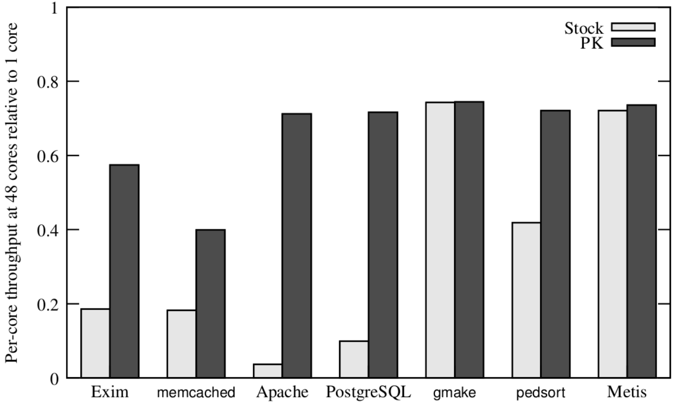
Scalable data structure.

Scalable lock.

An Analysis of Linux Scalability to Many Cores – OSDI'10

- Author : Silas Boyd-Wickizer, Austin T.Clements : MIT PDOS
- Problem : Linux Scalability
  - Linux Scalability의 문제 분석
- Solution : 벤치마크 구현 + 문제 해결
  - MOSBench 구현
  - Per-core data structure
  - Eliminating false sharing
  - Avoiding unneccessary locking
  - Sloppy counters
  - Multicore packet processing

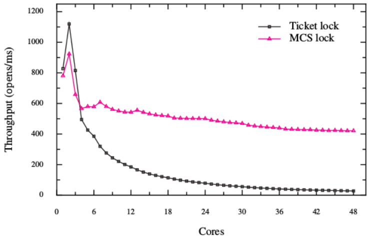
An Analysis of Linux Scalability to Many Cores – OSDI'10



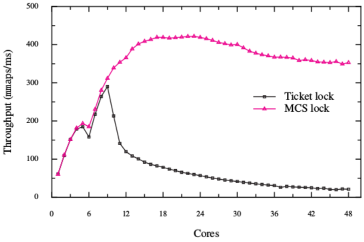
Non-scalable locks are dangerous - OLS 2012

- Author : Silas Boyd-Wickizer - MIT PODS
- Problem : Non-scalable locks
  - 리눅스가 Non-scalable locks 사용
- Solution : MCS lock을 사용하자
  - MCS lock을 리눅스에 구현 후 실험
  - 성능 향상

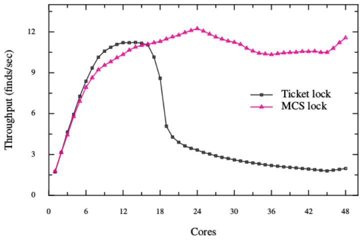
Non-scalable locks are dangerous - OLS 2012



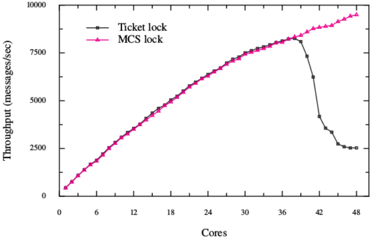
(a) Performance for FOPS.



(b) Performance for MEMPOP.



(c) Performance for PFIND.



(d) Performance for EXIM.

Scalable Address Spaces Using RCU Balanced Trees – ASPLOS'12

- Author : Austin T.Clements : MIT PDOS
- Problem : Contention Problem
  - mmap + munmap + page faults
- Solution : RCU 이용(BONSAI)
  - Linux Red-black tree
  - RCU balanced tree

Scalable Address Spaces Using RCU Balanced Trees – ASPLOS'12

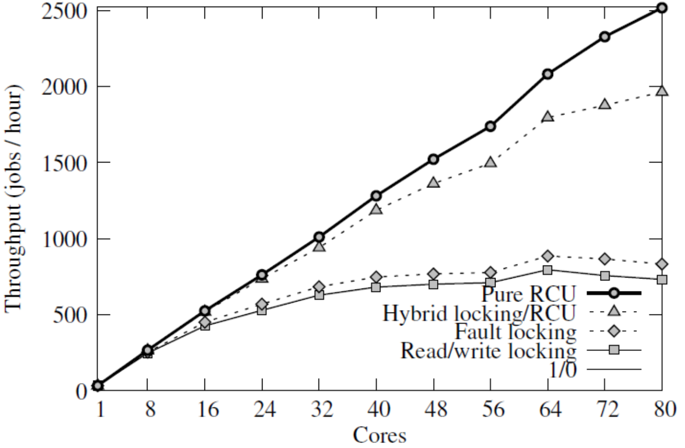


Figure 13. Metis throughput for each page fault concurrency design.

RadixVM: Scalable address spaces for multithreaded applications – EuroSYS'13

- Author : Austin T.Clements : MIT PDOS
- Problem : Contention Problem
  - mmap + munmap + page faults
- Solution : Lockless VM
  - TLB shutdown interrupt
  - Scalable reference counters

The Scalable Commutativity Rule-Designing Scalable Software for Multicore Processors – SOSP 13

- Author : Austin T.Clements : MIT PDOS
- Problem : 응용프로그램의 설계 문제
    - Scalable한 응용프로그램

Scalable data structure

Scalable operating system.

Scalable data structure.

Scalable lock.

Scalable data structure – stack

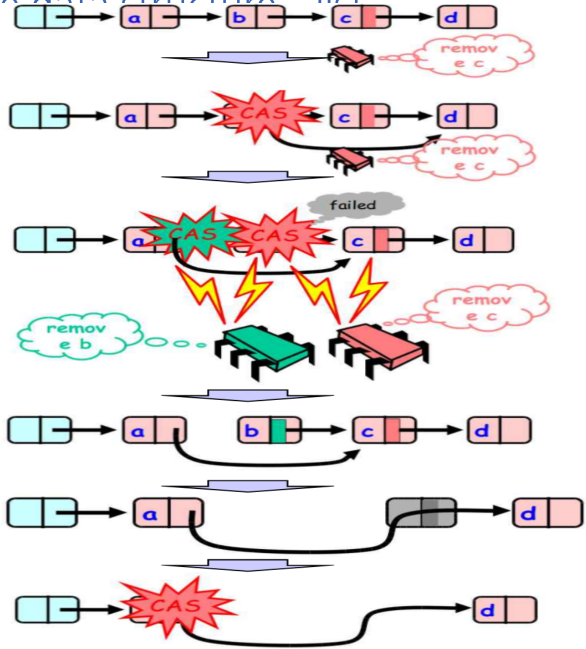
```
struct element {
    int key;
    int value;
    struct element *next;
  };

struct element *head;
```

```
struct element *pop(void) {
    again:
        struct element *e = head;
        if (cmpxchg(&head, e, e->next) != e)
            goto again;
        return e;
}
```

```
void push(struct element *e) {
    again:
        e->next = head;
        if (cmpxchg(&head, e->next, e) != e->next)
            goto again;
}
```

Scalable data structure – list



More than you ever wanted to know about synchronization: synchrobench, measuring the impact of the synchronization on concurrent algorithms – PPoPP 2015

- Author : Vincent Gramoli – NICTA
- 최신 알고르즘을 정리 또는 구현 후 공개소스로 공개
- 성능 측정

More than you ever wanted to know about synchronization

#	Algorithm	Ref.	Synchronization	Strategy	Authors	Data structures	Language
1	Practical binary tree	[7]	lock	optimistic	Stanford U.	binary tree	Java
2	Contention-friendly tree	[15]	lock	pessimistic	INRIA&NICTA&U. Sydney	binary tree	Java
3	Logical ordering tree	[22]	lock	pessimistic	Technion & ETHZ	binary tree	Java
4	Lock-free tree	[24]	read-modify-write	optimistic	Toronto U.&FORTH&York U.	binary tree	Java
5	Fast lock-free tree	[58]	read-modify-write	optimistic	U. of Texas, Dallas	binary tree	C/C++
6	Speculation-friendly tree	[14]	transaction	optimistic	EPFL & INRIA	binary tree	C/C++
7	Transactional red black tree	[8]	transaction	optimistic	Sun (Oracle)	binary tree	Java&C/C++
8	Citrus tree	[4]	read-copy-update	optimistic	Technion	binary tree	C/C++
9	j.u.c.copyOnWriteArraySet	[32]	copy-on-write	optimistic	Oracle	dynamic array	Java
10	java.util.Vector	[1]	lock	pessimistic	Oracle	dynamic array	Java
11	ReusableVector	[34]	transaction	optimistic	EPFL & U. Sydney	dynamic array	Java
12	j.u.c.ConcurrentHashMap	[49]	lock	pessimistic	SUNY	hash table	Java
13	Michael's hash table	[54]	read-modify-write	optimistic	IBM	hash table	C/C++
14	Cliff Click's hash map	[12]	read-modify-write	optimistic	Azul Systems	hash table	Java
15	Contention-friendly hash table	[13]	read-modify-write	optimistic	INRIA & EPFL	hash table	Java
16	Resizable hash table	[51]	read-modify-write	optimistic	Lehigh U. & Tianjin U.	hash table	Java
17	Elastic hash table	[26]	transaction	optimistic	EPFL & UniNE	hash table	Java&C/C++
18	Lazy linked list	[41]	lock	optimistic	Sun&Brown U.&Rochester U.	linked list	Java&C/C++
19	Lock-coupling linked list	[47]	lock	pessimistic	Brown U. & MIT	linked list	Java&C/C++
20	j.u.Collections.synchronizedSet	[32]	lock	pessimistic	Oracle	linked list	Java
21	Harris' linked list	[39]	read-modify-write	optimistic	Cambridge U.	linked list	Java&C/C++
22	Reusable linked list	[34]	transaction	optimistic	EPFL & U. Sydney	linked list	Java
23	Elastic linked list	[27]	transaction	optimistic	EPFL & UniNE	linked list	Java&C/C++
24	j.u.c.ConcurrentLinkedQueue	[56]	read-modify-write	optimistic	IBM & Rochester U.	queue	Java
25	ReusableLinkedQueue	[34]	transaction	optimistic	EPFL & U. Sydney	queue	Java
26	Optimistic skip list	[43]	lock	optimistic	Sun&Brown U.&Rochester U.	skip list	C/C++
27	Fraser skip list	[30]	read-modify-write	optimistic	Cambridge U.	skip list	C/C++
28	j.u.c.ConcurrentSkipListMap	[49]	read-modify-write	optimistic	SUNY	skip list	Java
29	No hot spot skip list	[16]	read-modify-write	optimistic	INRIA & U. Sydney	skip list	Java&C/C++
30	Rotating skip list	[21]	read-modify-write	optimistic	U. Sydney	skip list	C/C++
31	Elastic skip list	[26]	transaction	optimistic	EPFL & UniNE	skip list	Java&C/C++

Table 1. Algorithms of Synchrobench

Scalable lock

Scalable operating system.

Scalable data structure.

Scalable lock.

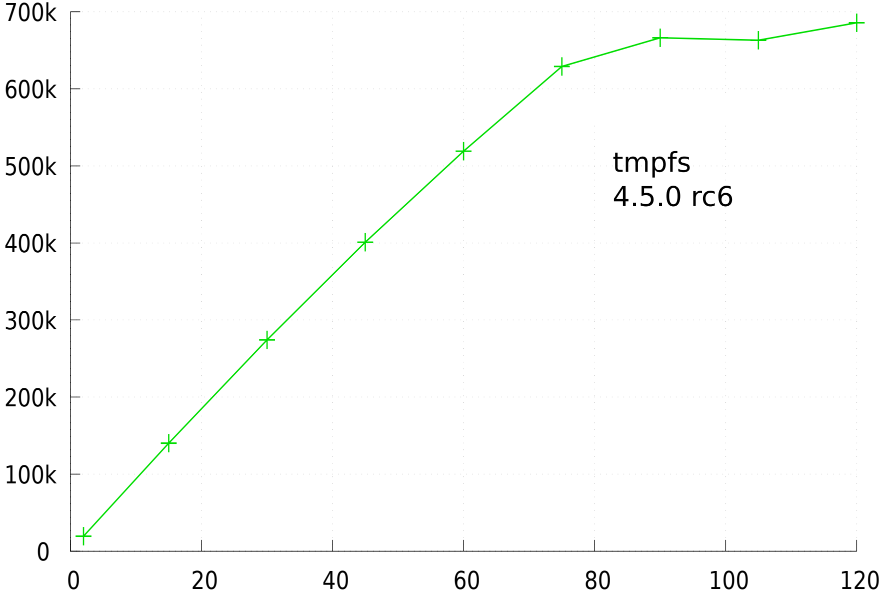
exclusive lock – spin lock

- test and set
- ticket lock
- Anderson Queue Lock
- MCS
- 등등등

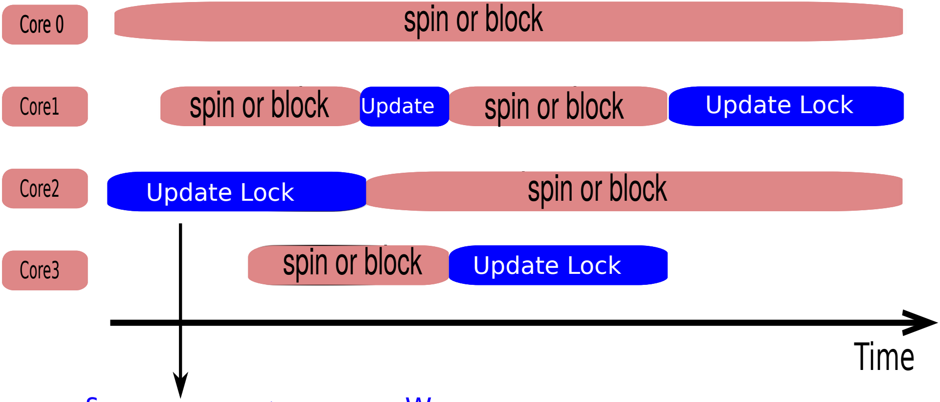
concurrent reads – read-mostly data structure

- read-write lock
- RCU
- RLU (SOSP ’15)

AIM7 Scalability – Linux 4.5.0



Concurrent updates



&anon_vma−>rwsem−W
&mapping−>i_mmap_rwsem-W

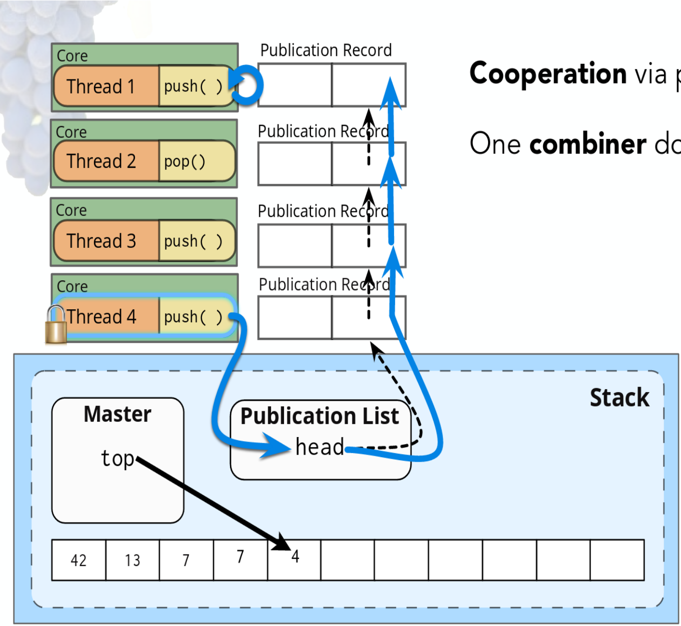
log-based concurrent updates

- FC(Flat Combining)
- Oplog
- LDU

Flat Combining

- Flat Combining and the Synchronization-Parallelism Tradeoff
- SPAA 2010

Flat Combining



Cooperation via publication list

One **combiner** does all the work

[1] *"Flat Combining and the Synchronization-Parallelism Tradeoff"*
Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir

Optimizing Communication Bottlenecks in Multiprocessor Operating System kernels – MIT Doctor thesis 2014

- Author : Silas Boyd-Wickizer – MIT PODS
- Problem : Update-heavy datastructure
  - Update-heavy한 상황
- Solution : OpLog
  - 새로운 lock 툴 제공
  - synchronized clocks(RDTSC and RDTSCP)을 이용
  - per-cpu로 데이터 저장 후 처음 read 발생 시 처리
  - path name looup, VM reverse mapping, and the state() system call

Optimizing Communication Bottlenecks in Multiprocessor Operating System kernels – MIT Doctor thesis 2014

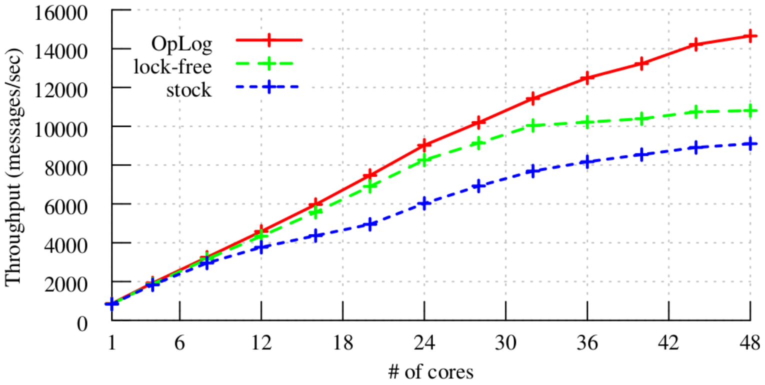
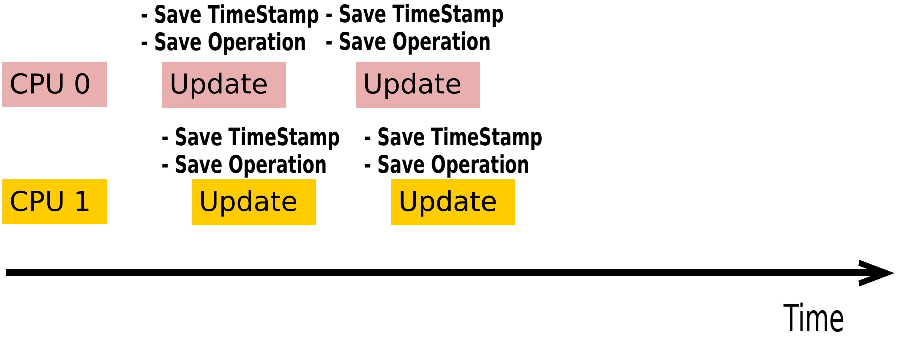
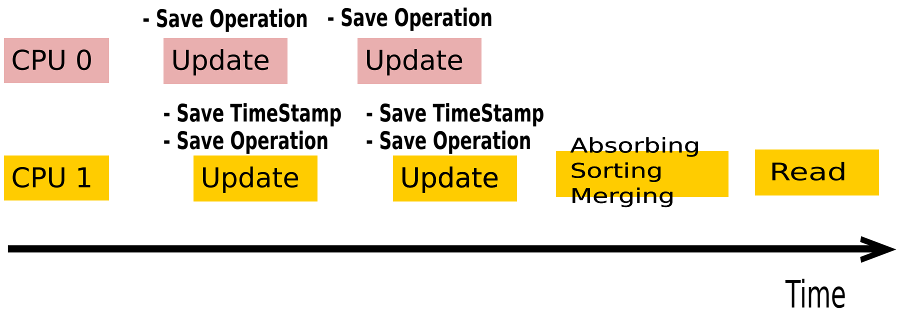


Figure 10: Performance of Exim.

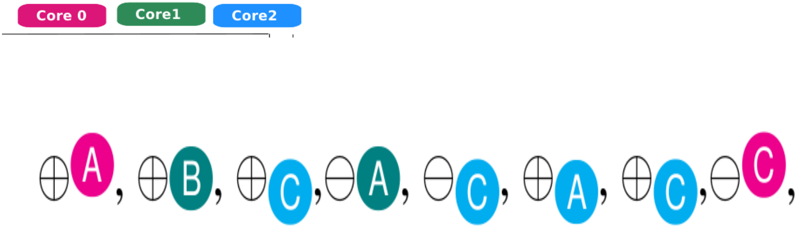
Concurrent updates – solution



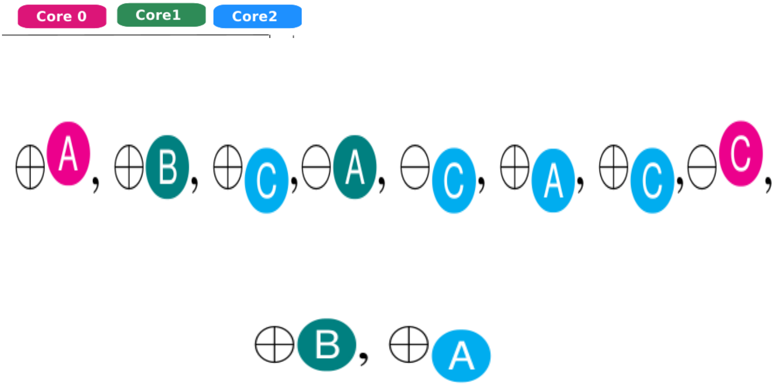
Concurrent updates – solution

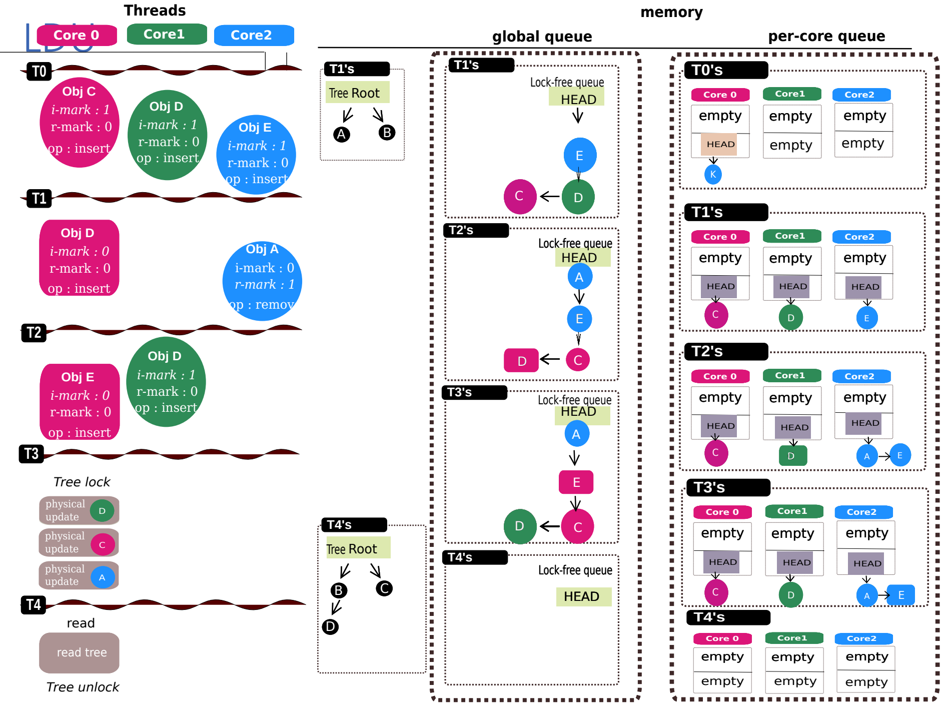


Synchronized timestamp counter



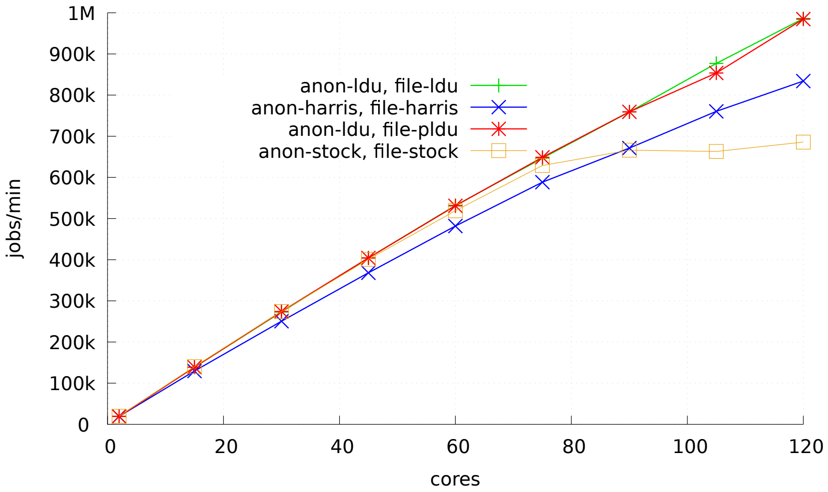
removing





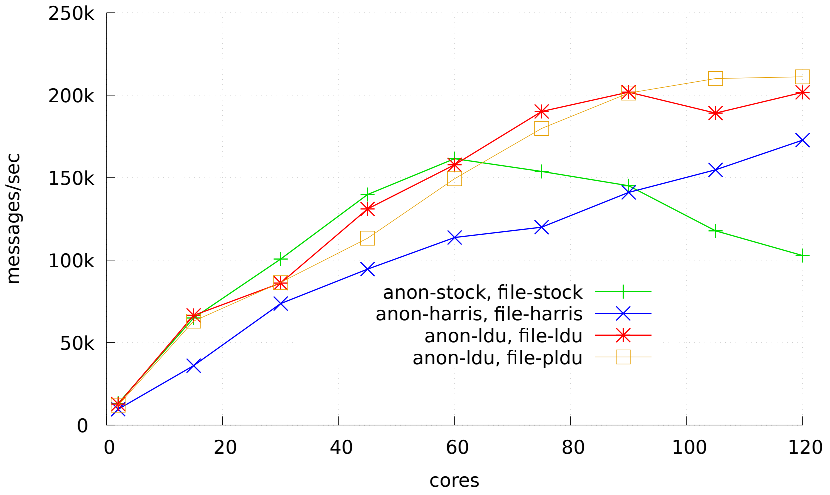
Threads
Core 0
Core1
Core2
memory
global queue
per-core queue
T0
Obj C
i-mark : 1
r-mark : 0
op : insert
Obj D
i-mark : 1
r-mark : 0
op : insert
Obj E
i-mark : 1
r-mark : 0
op : insert
T1
Obj D
i-mark : 0
r-mark : 0
op : insert
Obj A
i-mark : 0
r-mark : 1
op : remov
T2
Obj E
i-mark : 0
r-mark : 0
op : insert
Obj D
i-mark : 1
r-mark : 0
op : insert
T3
Tree lock
physical update
physical update
physical update
T4
read
read tree
Tree unlock
T1's
Tree Root
T4's
Tree Root
T1's
Lock-free queue
HEAD
T2's
Lock-free queue
HEAD
T3's
Lock-free queue
HEAD
T4's
Lock-free queue
HEAD
T0's
T1's
T2's
T3's
T4's
empty
HEAD

AIM7



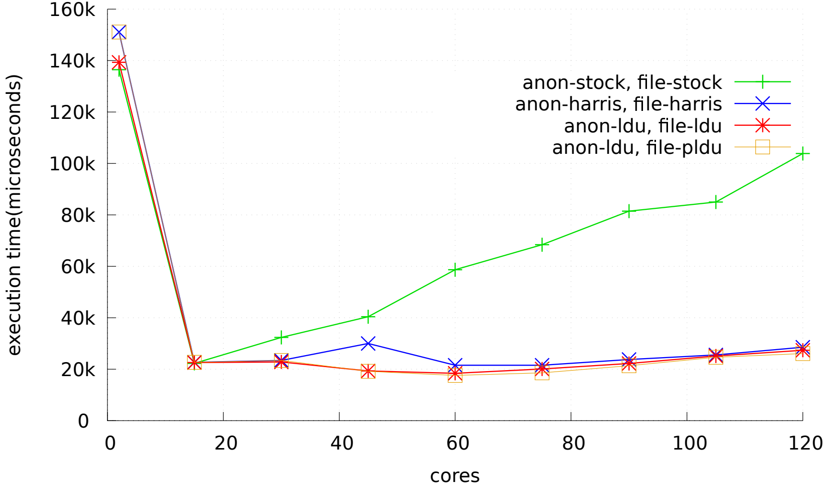
1M
900k
800k
700k
600k
500k
400k
300k
200k
100k
0
jobs/min
0
20
40
60
80
100
120
cores
anon-ldu, file-ldu
anon-harris, file-harris
anon-ldu, file-pldu
anon-stock, file-stock

exim



250k
200k
150k
100k
50k
0
messages/sec
0
20
40
60
80
100
120
cores
anon-stock, file-stock
anon-harris, file-harris
anon-ldu, file-ldu
anon-ldu, file-pldu

lmbench



anon-stock, file-stock
anon-harris, file-harris
anon-ldu, file-ldu
anon-ldu, file-pldu
execution time(microseconds)
160k
140k
120k
100k
80k
60k
40k
20k
0
0
20
40
60
80
100
120
cores

Summary

- Background of research
- LDU method and Evaluation
- Future plans and Summary
- https://github.com/KMU-embedded/scalablelinux