Scalable address spaces using RCU balanced trees

Austin T. Clements  M. Frans Kaashoek  Nickolai Zeldovich
MIT CSAIL
Motivation

Scalable address spaces using RCU balanced trees
Motivation

Scalable address spaces using RCU balanced trees

Multithreaded software
Motivation

Scalable address spaces using RCU balanced trees

Multithreaded software

Address spaces

OS kernel
Application scalability

Scalable address spaces using RCU balanced trees

Throughput (jobs / hour)

Cores

Metis [Boyd-Wickizer '10]

Linux 2.6.37

22x
Application scalability

Scalable address spaces using RCU balanced trees

Metis [Boyd-Wickizer '10]

Throughput (jobs / hour)

Cores

Linux 2.6.37

VM

22x
Application scalability

Metis [Boyd-Wickizer '10]

Throughput (jobs / hour) vs. Cores

- Idle: 45 s
- Kernel: 196 s
- User: 150 s
- Total: 391 s

Linux 2.6.37: 22x improvement

Scalable address spaces using RCU balanced trees
Application scalability

Metis [Boyd-Wickizer '10]

Throughput (jobs / hour)

Cores

Linux 2.6.37

Entering wait queues

idle 196 s

kernel 150 s

user 391 s

45 s

19%

22x
Application scalability

Metis [Boyd-Wickizer '10]

Scalable address spaces using RCU balanced trees

Throughput (jobs / hour)

Cores

Linux 2.6.37

Entering wait queues: 45 s
Atomic ops on shared data: 196 s
Kernel: 150 s
User: 391 s

Idle: 19%
Atomic ops: 62%

22x
Enable VM-intensive multithreaded software to scale.
Address spaces

Scalable address spaces using RCU balanced trees
Address spaces

mmap

munmap
Address spaces

mmap

page fault

munmap
Address spaces

Scalable address spaces using RCU balanced trees

mmap

page fault

munmap

VMA tree

Regions (VMA's)

Virtual memory

Page tables

Page directory

Scalable address spaces using RCU balanced trees
Address spaces

Scalable address spaces using RCU balanced trees

mmap

page fault

munmap

VMA tree  Regions (VMA's)  Virtual memory  Page tables  Page directory

Scalable address spaces using RCU balanced trees
Address spaces

Scalable address spaces using RCU balanced trees

mmap

page fault

munmap

VMA tree
Regions (VMA's)
Virtual memory
Page tables
Page directory

Scalable address spaces using RCU balanced trees
Address spaces

scalable address spaces using RCU balanced trees

mmap

page fault

munmap

VMA tree

Regions (VMA's)

Virtual memory

Page tables

Page directory

Scalable address spaces using RCU balanced trees
Address spaces

Scalable address spaces using RCU balanced trees

- **mmap**
  - Creates VMAs

- **page fault**

- **munmap**

VMA tree

Regions (VMA's)

Virtual memory

Page tables

Page directory

Scalable address spaces using RCU balanced trees
Address spaces

mmap
Creates VMAs

page fault

munmap

VMA tree

Regions (VMA's)

Virtual memory

Page tables

Page directory

Scalable address spaces using RCU balanced trees
Address spaces

- **mmap**
  - Creates VMAs
- **page fault**
  - Create and fill page table on demand
- **munmap**

VMA tree

Regions (VMA's)

Virtual memory

Page tables

Page directory

Scalable address spaces using RCU balanced trees
Address spaces

mmap
Creates VMAs

page fault
Create and fill page table on demand

munmap

VMA tree
Regions (VMA's)
Virtual memory
Page tables
Page directory

Scalable address spaces using RCU balanced trees
Address spaces

mmap
Creates VMAs

page fault
Create and fill page table on demand

munmap
Removes VMAs, clears page tables, frees pages

VMA tree
Regions (VMA's)
Virtual memory
Page tables
Page directory

Scalable address spaces using RCU balanced trees
Address spaces

mmap
- Creates VMAs

page fault
- Create and fill page table on demand

munmap
- Removes VMAs, clears page tables, frees pages

Two invariants:
- Internal consistency of VMA tree
- Consistency between VM representations

VMA tree

Regions (VMA's)

Virtual memory

Page tables

Page directory
Address spaces

mmap
Creates VMAs

page fault
Create and fill page table on demand

munmap
Removes VMAs, clears page tables, frees pages

VMA tree | Regions (VMA's) | Virtual memory | Page tables | Page directory

Scalable address spaces using RCU balanced trees
Address spaces

- **mmap** (W): Creates VMAs
- **page fault** (R): Create and fill page table on demand
- **munmap** (W): Removes VMAs, clears page tables, frees pages

VMA tree

- Regions (VMA's)
- Virtual memory
- Page tables
- Page directory

Scalable address spaces using RCU balanced trees
What's so bad about RW locks?

Parallel page faults,
What's so bad about RW locks?

Parallel page faults, mmap/munmap delayed

Scalable address spaces using RCU balanced trees
What's so bad about RW locks?

Parallel page faults, mmap/munmap delayed

Serial mmap/munmap,
What's so bad about RW locks?

- Parallel page faults, mmap/munmap delayed
- Serial mmap/munmap, page faults delayed
- Parallel page faults
What's so bad about RW locks?
What's so bad about RW locks?

Limited concurrency
What's so bad about RW locks?

- Limited concurrency
- Increasing lock cache line movement

Scalable address spaces using RCU balanced trees
What's so bad about RW locks?

Must minimize cache line movement.
Page faults must run concurrently with mapping operations.

Scalable address spaces using RCU balanced trees
What's so bad about RW locks?

Must minimize cache line movement.
Page faults must run concurrently with mapping operations.

Scalable address spaces using RCU balanced trees
What's so bad about RW locks?

Must minimize cache line movement.
Page faults must run concurrently with mapping operations.

Scalable address spaces using RCU balanced trees
Contributions

An RCU-based scheme for highly scalable page fault handling

An RCU-compatible balanced binary tree (Bonsai)

Implementation and evaluation in Linux 2.6.37
A Read-Copy-Update primer
A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free
A Read-Copy-Update primer

Lock-free reads  +  Single pointer update  +  Delayed free

head
A Read-Copy-Update primer

Lock-free reads  +  Single pointer update  +  Delayed free

head

length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
A Read-Copy-Update primer

Lock-free reads  +  Single pointer update  + Delayed free

head

length()

```c
p = head;
for (i=0; p; p=p->next, i++);
return i;
```

No locks, no barriers
A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free

```
length()
p = head;
for (i=0; p; p=p->next, i++);
return i;
```

```
pop_n(n)
lock();
for (p=head; p&&n; p=p->next, n--)
rcu_free(p);
head = p;
unlock();
```

No locks, no barriers
A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free

**length()**

```c
p = head;
for (i=0; p; p=p->next, i++);
return i;
```

No locks, no barriers

**pop_n(n)**

```c
lock();
for (p=head; p&&n; p=p->next, n--)
    rcu_free(p);
head = p;
unlock();
```

Updates exactly one pointer
A Read-Copy-Update primer

Lock-free reads  +  Single pointer update  +  Delayed free

head

length()

\( p = \text{head}; \)
\( \text{for} \ (i=0; \ p; \ p=p->\text{next}, \ i++); \)
\( \text{return} \ i; \)

No locks, no barriers

pop_n(n)
\( \text{lock}(); \)
\( \text{for} \ (p=\text{head}; \ p&&n; \ p=p->\text{next}, \ n--;) \)
\( \text{rcu}\_\text{free}(p); \)
\( \text{head} = p; \)
\( \text{unlock}(); \)

Updates exactly one pointer
A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free

head

length()

\[
p = \text{head};
\]
\[
\text{for (i=0; } p; p=p->\text{next, } i++);
\]
\[
\text{return } i;
\]

No locks, no barriers

Updates exactly one pointer

pop_n(n)

\[
\text{lock();}
\]
\[
\text{for (p=\text{head}; p&&n; p=p->\text{next, } n--)}
\]
\[
\text{rcu_free(p);}
\]

head = p;

unlock();
A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free

No locks, no barriers

Updates exactly one pointer
A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free

length()

p = head;
for (i=0; p; p=p->next, i++);
return i;

No locks, no barriers

Updates exactly one pointer

pop_n(n)
lock();
for (p=head; p&&n; p=p->next, n--)
rcu_free(p);
head = p;
unlock();

Scalable address spaces using RCU balanced trees
A Read-Copy-Update primer

Lock-free reads  +  Single pointer update  +  Delayed free

head

length()

pop_n(n)

No locks, no barriers

Updates exactly one pointer
A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free

head

length()
p = head;
for (i=0; p; p=p->next, i++); return i;

No locks, no barriers

pop_n(n)
lock();
for (p=head; p&&n; p=p->next, n--)
rcu_free(p);
head = p;
unlock();

Updates exactly one pointer
A Read-Copy-Update primer

锁无锁读 + 单独指针更新 + 延迟释放

长度(

p = head;
for (i=0; p; p=p->next, i++);
return i;

无锁，无障碍物

pop_n(n)
lock();
for (p=head; p&&n; p=p->next, n--)
rcu_free(p);
head = p;
unlock();

更新恰好一个指针
A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free

head

length()

p = head;
for (i=0; p; p=p->next, i++);
return i;

No locks, no barriers

pop_n(n)
lock();
for (p=head; p&&n; p=p->next, n--)
rcu_free(p);
head = p;
unlock();

Updates exactly one pointer
A Read-Copy-Update primer

Lock-free reads  +  Single pointer update  +  Delayed free

head

length()

No locks, no barriers

p = head;
for (i=0; p; p=p->next, i++);
return i;

pop_n(n)
lock();
for (p=head; p&&n; p=p->next, n--)
rcu_free(p);
head = p;
unlock();

Updates exactly one pointer

Free is delayed until all readers return (e.g., by waiting for all CPU's to schedule)

Scalable address spaces using RCU balanced trees
A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free

head

length()

\[ p = \text{head}; \]
\[ \text{for} \ (i=0; \ p; \ p=p->\text{next}, \ i++); \]
\[ \text{return} \ i; \]

No locks, no barriers

pop_n(n)

\[ \text{lock}(); \]
\[ \text{for} \ (p=\text{head}; \ p&&n; \ p=p->\text{next}, \ n--); \]
\[ \text{rcu\_free}(p); \]
\[ \text{head} = p; \]
\[ \text{unlock}(); \]

Updates exactly one pointer

Free is delayed until all readers return (e.g., by waiting for all CPU's to schedule)
RCU is already used pervasively in Linux
RCU challenges

RCU is already used pervasively in Linux
... except in the VM system
RCU challenges

RCU is already used pervasively in Linux
... except in the VM system

Address space operations update many pointers

The page fault handler isn't read-only
General approach

- Remove address space lock from page fault fast path
- Reduce writes required to map/unmap memory
- Detect remaining races and retry racing page faults
General approach

Scalable address spaces using RCU balanced trees

VMA tree  Regions (VMA's)  Virtual memory  Page tables  Page directory
General approach

Scalable address spaces using RCU balanced trees

New data structure

VMA tree

Regions (VMA's)

Virtual memory

Page tables

Page directory

Scalable address spaces using RCU balanced trees
General approach

New data structure

VMA tree
Regions (VMA's)
Virtual memory
Page tables
Page directory

Race detection and recovery

Scalable address spaces using RCU balanced trees
The VMA tree problem

Tree balancing is not RCU-friendly
Tree balancing is not RCU-friendly
The VMA tree problem

Tree balancing is not RCU-friendly
The VMA tree problem

Tree balancing is not RCU-friendly
The VMA tree problem

Tree balancing is not RCU-friendly
The VMA tree problem

Tree balancing is not RCU-friendly

Balancing performs multiple pointer updates, so lock-free readers can see partial updates [Fraser '03, Fraser '07, Howard '10, etc]
The basic Bonsai tree
Functional data structures don't update any pointers!
"Modification" creates a new structure
The basic Bonsai tree

Functional data structures don't update any pointers! "Modification" creates a new structure
The basic Bonsai tree

Functional data structures don't update any pointers! "Modification" creates a new structure
Functional data structures don't update any pointers! "Modification" creates a new structure.
The basic Bonsai tree

Functional data structures don't update any pointers! "Modification" creates a new structure

But we don't want a new tree
The basic Bonsai tree

Functional data structures don't update any pointers! "Modification" creates a new structure.

Path-copied to root

New, rotated subtree

But we don't want a new tree
The basic Bonsai tree

Functional data structures don't update any pointers! "Modification" creates a new structure

Single pointer update!

Path-copied to root

New, rotated subtree

But we don't want a new tree
The basic Bonsai tree

Functional data structures don't update any pointers! "Modification" creates a new structure.

Single pointer update!

Path-copied to root

New, rotated subtree

We don't need the old tree

But we don't want a new tree
Functional data structures don't update any pointers! "Modification" creates a new structure

We don't need the old tree

But we don't want a new tree
The better Bonsai tree

We can do better

\[ O(n) \]

\[ O(n) \]
We can do better
Insertion allocates $O(\log n)$ nodes, frees $O(\log n)$ nodes
The better Bonsai tree

We can do better

Insertion allocates $O(\log n)$ nodes, frees $O(\log n)$ nodes

Path copying crucial for referential transparency, wasted work for us
We can do better
Insertion allocates $O(\log n)$ nodes, frees $O(\log n)$ nodes

Path copying crucial for referential transparency, wasted work for us

Rotation still out-of-place
The better Bonsai tree

We can do better
Insertion allocates $O(\log n)$ nodes, frees $O(\log n)$ nodes

Push pointer update down

Rotation still out-of-place
We can do better
Insertion allocates $O(\log n)$ nodes, frees $O(\log n)$ nodes

Push pointer update down

Multiple rotations commit separately; lock-free lookup is still safe

Rotation still out-of-place
The better Bonsai tree

We can do better
Insertion allocates $O(\log n)$ nodes, frees $O(\log n)$ nodes

Multiple rotations commit separately; lock-free lookup is still safe

Push pointer update down

Rotation still out-of-place

Scalable address spaces using RCU balanced trees
The better Bonsai tree

We can do better
Insertion allocates $O(\log n)$ nodes, frees $O(\log n)$ nodes

~2

~1

Bonsai maintains internal tree consistency while enabling lock-free VMA lookup

Multiple rotations commit separately; lock-free lookup is still safe

Rotation still out-of-place

Push pointer update down
Page table race

page_fault(va)
▷ Find VMA for va
  ▷ Allocate page table as needed
  ▷ Allocate page and set PTE
Page table race

page_fault(va)
  ▶ Find VMA for va
  ▶ Walk page table tree
    ▶ Allocate page table as needed
  ▶ Allocate page and set PTE
Page table race

page_fault(va)
▷ Find VMA for va
▷ Walk page table tree
  ▷ Allocate page table as needed
▷ Allocate page and set PTE
page_fault(va)
▷ Find VMA for va
▷ Walk page table tree
  ▷ Allocate page table as needed
▷ Allocate page and set PTE
page_fault(va)
▷ Find VMA for va
▷ Walk page table tree
  ▷ Allocate page table as needed
▷ Allocate page and set PTE
page_fault(va)
▷ Find VMA for va
▷ Walk page table tree
  ▷ Allocate page table as needed
▷ Allocate page and set PTE
Page table race

page_fault(va)
  ▶ Find VMA for va
  ▶ Walk page table tree
    ▶ Allocate page table as needed
  ▶ Allocate page and set PTE

Previously protected by the AS lock; now lock-free
Page table race

page_fault(va)
▶ Find VMA for va
▶ Walk page table tree
  ▶ Allocate page table as needed
▶ Allocate page and set PTE

munmap(start, end)
▶ Removes/adjusts VMA's
▶ Clear page table entries and free pages
▶ Free unused page tables
 Previously protected by the AS lock; now lock-free

Page table race

```plaintext
page_fault(va)
 ▶ Find VMA for va
 ▶ Walk page table tree
 ▶ Allocate page table as needed
 ▶ Allocate page and set PTE

munmap(start, end)
 ▶ Removes/adjusts VMA's
 ▶ Clear page table entries and free pages
 ▶ Free unused page tables
```
Page table race

page_fault(va)
▷ Find VMA for va
▷ Walk page table tree
  ▷ Allocate page table as needed
▷ Allocate page and set PTE

munmap(start, end)
▷ Removes/adjusts VMA's
▷ Clear page table entries and free pages
▷ Free unused page tables

Previously protected by the AS lock; now lock-free
Previously protected by the AS lock; now lock-free

```plaintext
page_fault(va)
▷ Find VMA for va
▷ Walk page table tree
   ▷ Allocate page table as needed
▷ Allocate page and set PTE

munmap(start, end)
▷ Removes/adjusts VMA's
▷ Clear page table entries and free pages
▷ Free unused page tables
```
Page table race

page_fault(va)
▷ Find VMA for va
▷ Walk page table tree
  ▷ Allocate page table as needed
▷ Allocate page and set PTE

munmap(start, end)
▷ Removes/adjusts VMA's
▷ Clear page table entries and free pages
▷ Free unused page tables

Previously protected by the AS lock; now lock-free
Page table race

Previously protected by the AS lock; now lock-free

page_fault(va)
▷ Find VMA for va
▷ Walk page table tree
  ▷ Allocate page table as needed
▷ Allocate page and set PTE

munmap(start, end)
▷ Removes/adjusts VMA's
▷ Clear page table entries and free pages
▷ Free unused page tables
Page table race

page_fault(va)
➤ Find VMA for va
➤ Walk page table tree
    ➤ Allocate page table as needed
➤ Allocate page and set PTE

munmap(start, end)
➤ Removes/adjusts VMA's
➤ Clear page table entries and free pages
➤ Free unused page tables

Previously protected by the AS lock; now lock-free
Page table race

page_fault(va)
- Find VMA for va
- Walk page table tree
  - Allocate page table as needed
- Allocate page and set PTE

munmap(start, end)
- Removes/adjusts VMA's
- Clear page table entries and free pages
- RCU-Free unused page tables

Previously protected by the AS lock; now lock-free

Makes lock-free walk safe
Page table race

page_fault(va)
   ▶ Find VMA for va
   ▶ Walk page table tree
      ▶ Allocate page table as needed
   ▶ Allocate page and set PTE

munmap(start, end)
   ▶ Removes/adjusts VMA's
   ▶ Clear page table entries and free pages
   ▶ RCU-Free unused page tables

Makes lock-free walk safe

Previously protected by the AS lock; now lock-free

Oops
page_fault(va)
▷ Find VMA for va
▷ Walk page table tree
   ▷ Allocate page table as needed
   ▷ If VMA unlinked or va invalid, retry fault
▷ Allocate page and set PTE

munmap(start, end)
▷ Removes/adjusts VMA's
▷ Clear page table entries and free pages
▷ RCU-Free unused page tables

Previously protected by the AS lock; now lock-free

Makes lock-free walk safe
Page table race

page_fault(va)
▷ Find VMA for va
▷ Walk page table tree
   ▷ Allocate page table as needed
   ▷ If VMA unlinked or va invalid, retry fault
▷ Allocate page and set PTE

munmap(start, end)
▷ Removes/adjusts VMA's
▷ Clear page table entries and free pages
▷ RCU-Free unused page tables

Previously protected by the AS lock; now lock-free

Makes lock-free walk safe

RCU + race detection + retries is a potent combination.
Scalable address spaces using RCU balanced trees
Evaluation

How much does our design help?
Evaluation

How much does our design help?

Is this just a stop-gap solution?
Approach

Implemented in Linux 2.6.37
Approach

Scalable address spaces using RCU balanced trees

Implemented in Linux 2.6.37

Psearchy

Metis

Dedup

Scalable address spaces using RCU balanced trees
Approach

Scalable address spaces using RCU balanced trees

Implemented in Linux 2.6.37

Psearchy

Metis

Dedup

Scalable address spaces using RCU balanced trees
Scalable address spaces using RCU balanced trees

Implemented in Linux 2.6.37

Psearchy

Metis

Dedup

80 core Intel Xeon

Scalable address spaces using RCU balanced trees
Application scalability (Metis)

Scalable address spaces using RCU balanced trees

Scalable address spaces using RCU balanced trees
Application scalability (Metis)

Scalable address spaces using RCU balanced trees

![Graph showing throughput vs cores for Scalable VM and Stock Linux, with Scalable VM having a throughput of 76x at 80 cores and Stock Linux having a throughput of 22x at 80 cores.]

Scalable address spaces using RCU balanced trees
Application scalability (Metis)

Scalable address spaces using RCU balanced trees

Throughput (jobs / hour)

Cores

Stock Linux

Scalable VM

45 s

idle

196 s

kernel

150 s

user

102 s

114 s

11 s

76x

22x

Scalable address spaces using RCU balanced trees
Application scalability (Psearchy, dedup)

Scalable address spaces using RCU balanced trees

**Psearchy**

Throughput (jobs / hour) vs. Cores

- Scalable VM (green)
- Stock Linux (blue)

**Dedup**

Throughput (jobs / hour) vs. Cores

- Scalable VM (green)
- Stock Linux (blue)
Application scalability (Psearchy, dedup)

Address space scalability matters for real applications. Our design eliminates kernel bottlenecks.
Address space scalability matters for real applications. Our design eliminates kernel bottlenecks.

Metis, Psearchy and Dedup are >80% user space
How does the VM system perform in isolation?
Microbenchmark design

250 GB
Microbenchmark design

Scalable address spaces using RCU balanced trees

250 GB

Page fault

Page fault

Page fault
Microbenchmark design

250 GB

Page fault
Page fault
Page fault

mmap
munmap

Sleep
Microbenchmark design

Scalable address spaces using RCU balanced trees

250 GB

Page fault → Page fault → Page fault

page fault load

mmap load

{mmap load

Sleep

mmap munmap
Page faults have constant cost

Page fault rate, no mmap

Page faults/sec (millions)

Cores

Scalable VM
Stock Linux
Page faults have constant cost

Scalable address spaces using RCU balanced trees

Page fault rate, no mmap

Page faults/sec (millions)

Cores

Metis

Scalable VM
Stock Linux
Page faults have constant cost

Scalable address spaces using RCU balanced trees

Page faults/sec (millions)

Fraction of time in mmap/munmap

Scalable VM

Stock Linux

Cores

Page faults/sec (millions)

0%

20%

40%

60%

80%

100%
Page faults have constant cost.

Scalable address spaces using RCU balanced trees.
Page faults have constant cost

![Graph showing page fault rate, no mmap]

- **Scalable VM**
  - Page faults/sec (millions): 20
  - 9269 cycles

- **Stock Linux**
  - Page faults/sec (millions): 0
  - 320,818 cycles

Scalable address spaces using RCU balanced trees
Page faults have constant cost

Page fault rate, no mmap

Page faults can scale at constant cost to very high rates and regardless of memory mapping load.
Related work

RCU [McKenney '98, '01, '04, '06]

Data structures
• RCU [Fraser '03, '07, Howard '10]
• Concurrent [Lehman '81, Pugh '90]
• Functional [Nievergelt '72, Adams '92]

Multicore benchmarks [Boyd-Wickizer '10, Bienia '11]
22x → 76x

Address space scalability matters
22x → 76x

Address space scalability matters

With new data structures like Bonsai, page faults can shed coarse-gained locks
22x → 76x

Address space scalability matters

With new data structures like Bonsai, page faults can shed coarse-gained locks

Page faults can scale at constant cost
Conclusion

Address space scalability matters

With new data structures like Bonsai, page faults can shed coarse-grained locks

Page faults can scale at constant cost

Thank you