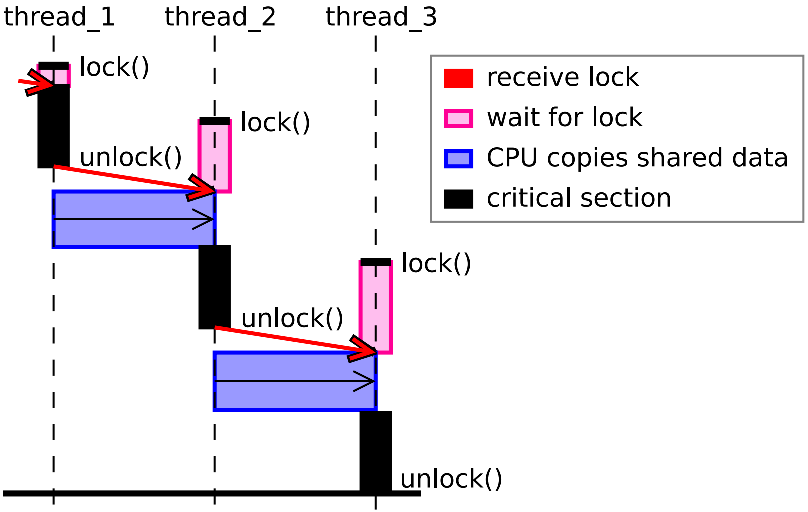
Remote Core Locking:

Performance through Serialisation

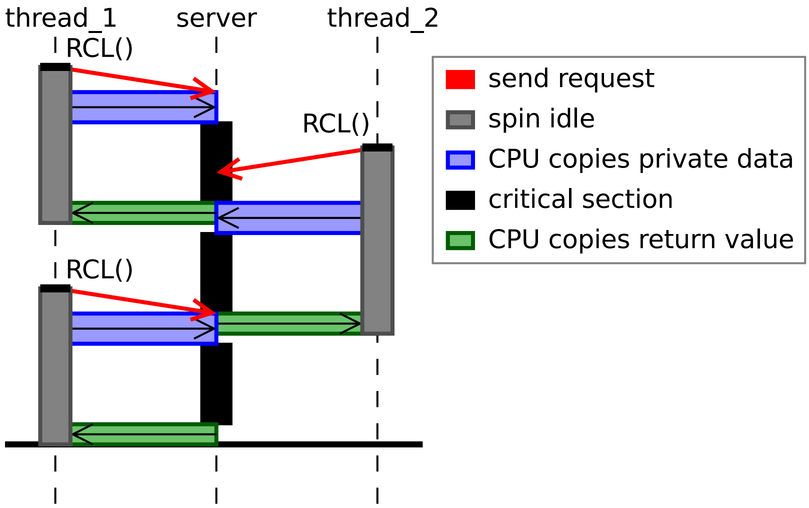
Michael Gebhard

2017-02-07

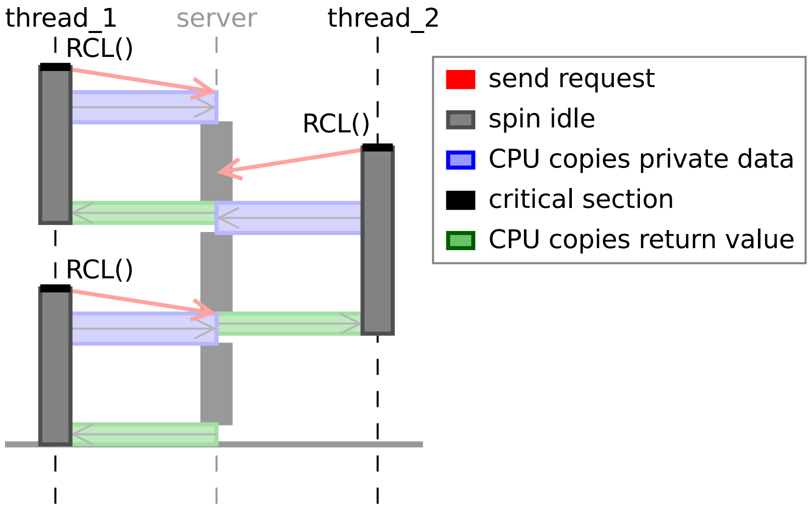
Ordinary Locks



RCL Locks


thread_1
server
thread_2
RCL()
RCL()
RCL()
🟥 send request
▨ spin idle
🟦 CPU copies private data
⬛ critical section
🟩 CPU copies return value

Client



thread_1
server
thread_2
RCL()
RCL()
RCL()
🟥 send request
▨ spin idle
🟦 CPU copies private data
■ critical section
🟩 CPU copies return value

Client Transformation

Ordinary

```
function(...) {
    [...]
    lock()
    //critical section
    [...]
    unlock()
    [...]
}
```

Client Transformation

Ordinary

```
function(...) {
    [...]
    lock()
    //critical section
    [...]
    unlock()
    [...]
}
```

RCL

```
function(...) {
    [...]
    RCL_call(&lock,
        &context,
        &crit_sec_1)
    [...]
}

//critical section
crit_sec_1(context) {
    [...]
}
```

Client Transformation

Ordinary

```
function(...) {
    [...]
    lock()
    //critical section
    if (condition) {
        unlock()
        [..path 1]
        return
    }
    [...]
    unlock()
    [..path 2]
}
```

RCL

```
function(...) {
    [...]
    RCL_call(&lock,
        &context,
        &crit_sec_1)
    [...]
}

//critical section
crit_sec_1(context) {
    [...]
}
```

Client Transformation

Ordinary

```
function(...) {
    [...]
    lock()
    //critical section
    if (condition) {
        unlock()
        [..path 1]
        return
    }
    [...]
    unlock()
    [..path 2]
}
```

RCL

```
function(...) {
    [...]
    RCL_call(&lock,
        &context,
        &crit_sec_1)
    [...]
}

//critical section
crit_sec_1(context) {
    if (condition) {
        return 1
    }
    [...]
    return 2
}
```

Client Transformation

Ordinary

```
function(...) {
    [...]
    lock()
    //critical section
    if (condition) {
        unlock()
        [..path 1]
        return
    }
    [...]
    unlock()
    [..path 2]
}
```

RCL

```
function(...) {
    [...]
    ret = RCL_call(&lock,
        &context,
        &crit_sec_1)
    switch ret
        case 1:
            [..path 1]
        case 2:
            [..path 2]
}

//critical section
crit_sec_1(context) {
    if (condition) {
        return 1
    }
    [...]
    return 2
}
```

Client Transformation

Ordinary

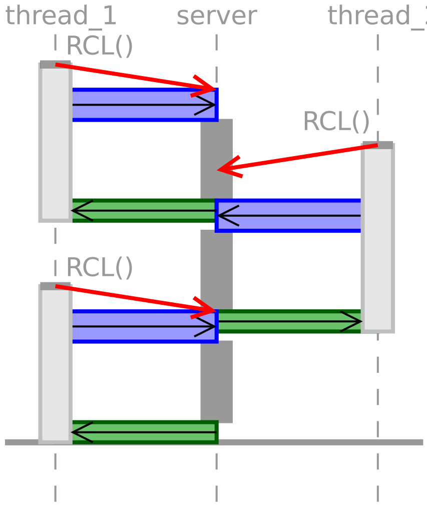
```
function(...) {
    [...]
    lock()
    //critical section
    if (condition) {
        unlock()
        [..path 1]
        return
    }
    [...]
    unlock()
    [..path 2]
}
```

RCL

```
function(...) {
    [...]
    ret = RCL_call(&lock,
        &context,
        &crit_sec_1)
    switch ret
        case 1:
            [..path 1]
        case 2:
            [..path 2]
}

//critical section
crit_sec_1(context) {
    if (condition) {
        return 1
    }
    [...]
    return 2
}
```

Client-Server-Communication



🟥	send request
▩	spin idle
🟦	CPU copies private data
■	critical section
🟩	CPU copies return value

Client-Server-Communication

client server

Client-Server-Communication

client 1 cache

server cache

Client-Server-Communication

client 1 cache

lock	context	function
&lock 1	0xBEEF	NULL

server cache

lock	context	function	
&lock 1	0xBEEF	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

Client-Server-Communication

client 1 cache

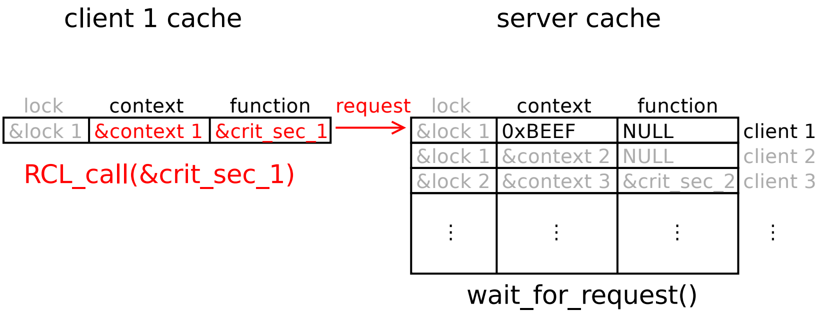
lock	context	function
&lock 1	0xBEEF	NULL

server cache

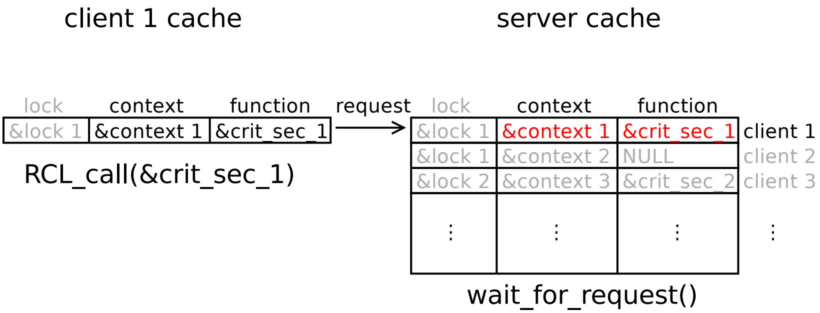
lock	context	function	
&lock 1	0xBEEF	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

wait_for_request()

Client-Server-Communication



Client-Server-Communication



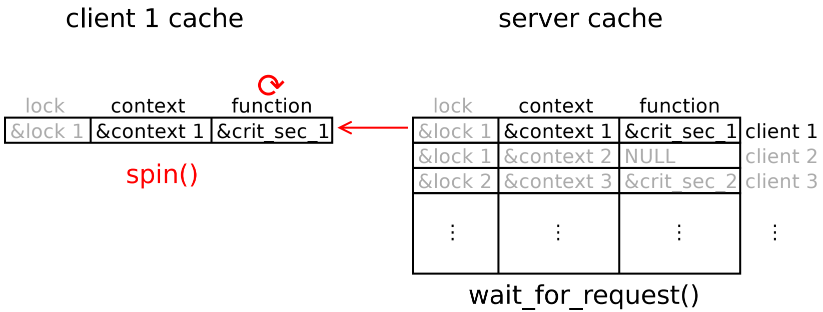
client 1 cache:

lock	context	function
&lock 1	&context 1	&crit_sec_1

server cache:

lock	context	function	
&lock 1	&context 1	&crit_sec_1	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

Client-Server-Communication



Client-Server-Communication

client 1 cache

⟳

lock	context	function
&lock 1	&context 1	&crit_sec_1

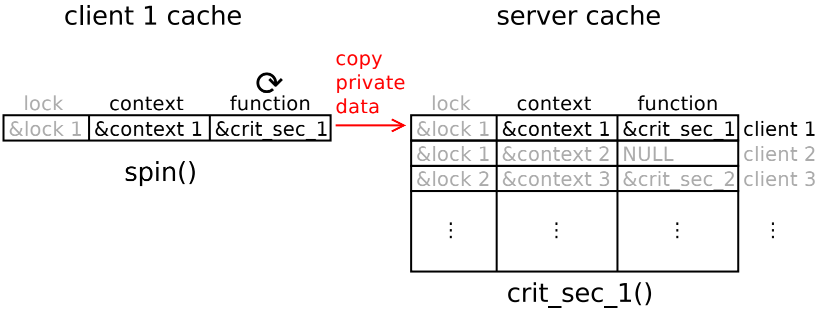
spin()

server cache

lock	context	function	
&lock 1	&context 1	&crit_sec_1	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

crit_sec_1()

Client-Server-Communication



Client-Server-Communication

client 1 cache

⟳

lock	context	function
&lock 1	&context 1	&crit_sec_1

spin()

server cache

lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

crit_sec_1()

Client-Server-Communication

client 1 cache

lock	context	⟳ function
&lock 1	&context 1	&crit_sec_1

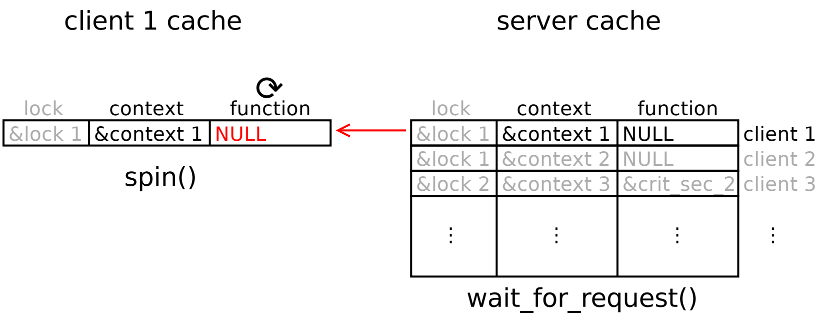
spin()

server cache

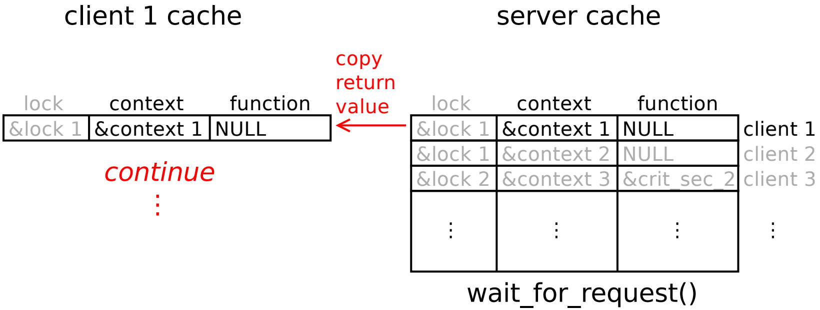
lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

wait_for_request()

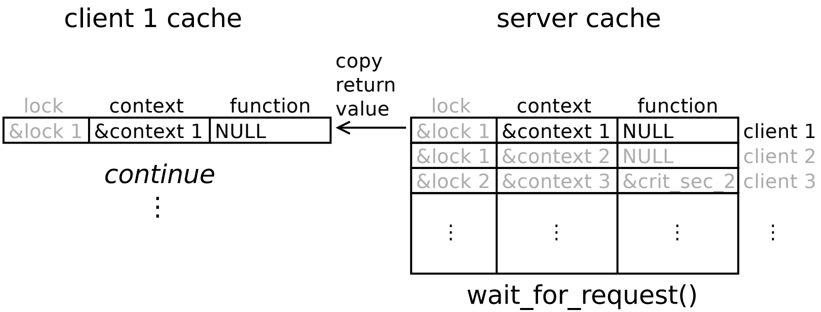
Client-Server-Communication



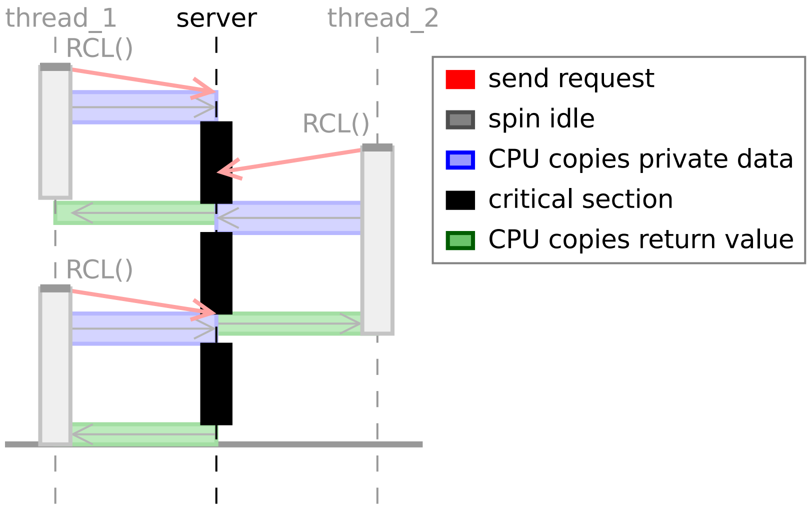
Client-Server-Communication



Client-Server-Communication



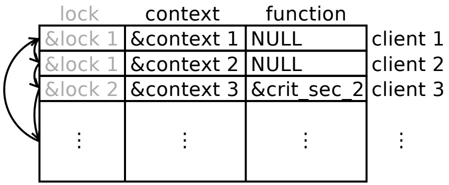
Server


thread_1
server
thread_2
RCL()
RCL()
RCL()
🟥 send request
▨ spin idle
🟦 CPU copies private data
■ critical section
🟩 CPU copies return value

Servicing Thread

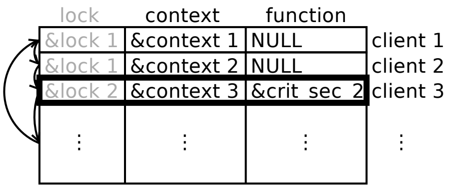
lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

Servicing Thread



lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

Servicing Thread



lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

Servicing Thread

lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

&context_3

Servicing Thread

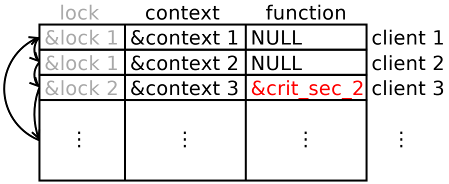
lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

crit_sec_2(&context_3)

Servicing Thread

lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	NULL	client 3
⋮	⋮	⋮	⋮

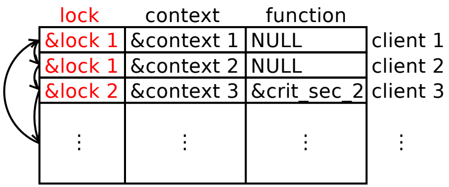
Servicing Thread



lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

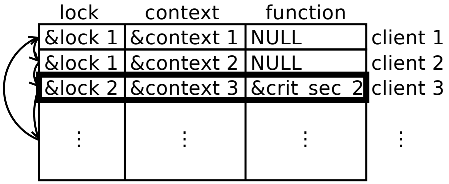
crit_sec_2(&context_3)

Servicing Thread



	lock	context	function	
	&lock 1	&context 1	NULL	client 1
	&lock 1	&context 2	NULL	client 2
	&lock 2	&context 3	&crit_sec_2	client 3
	⋮	⋮	⋮	⋮

Servicing Thread



lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

Servicing Thread

lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

lock(&lock_2)

Servicing Thread

lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

&context_3

Servicing Thread

lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	&crit_sec_2	client 3
⋮	⋮	⋮	⋮

crit_sec_2(&context_3)

Servicing Thread

lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	NULL	client 3
⋮	⋮	⋮	⋮

Servicing Thread

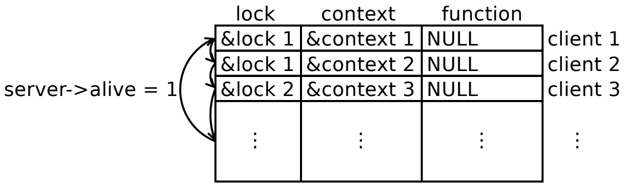
lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	NULL	client 3
⋮	⋮	⋮	⋮

unlock(&lock_2)

Servicing Thread

lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	NULL	client 3
⋮	⋮	⋮	⋮

Servicing Thread



lock	context	function	
&lock 1	&context 1	NULL	client 1
&lock 1	&context 2	NULL	client 2
&lock 2	&context 3	NULL	client 3
⋮	⋮	⋮	⋮

Management

Management Thread

```
if (server->alive == 0) {
    new servicing_thread()
} else {
    await(timeout)
}
```

Management

Management Thread

```
if (server->alive == 0) {
    new servicing_thread()
} else {
    await(timeout)
}
```

Management

Management Thread

```
if (server->alive == 0) {
    new servicing_thread()
} else {
    await(timeout)
}
```

Backup Thread

```
server->alive = 0
wakeup(management_thread)
```

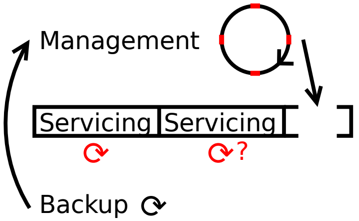
Management

Management Thread

```
if (server->alive == 0) {
    new servicing_thread()
} else {
    await(timeout)
}
```

Backup Thread

```
server->alive = 0
wakeup(management_thread)
```



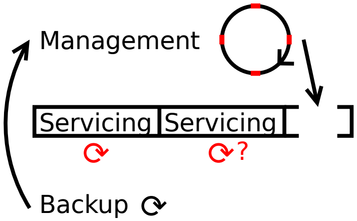
Management

Management Thread

```
if (server->alive == 0) {
    new servicing_thread()
} else {
    await(timeout)
}
```

Backup Thread

```
server->alive = 0
wakeup(management_thread)
```



Scheduling

Policy:

POSIX FIFO

Priorities:

4	Management Thread
3	Servicing Threads
2	Backup Thread

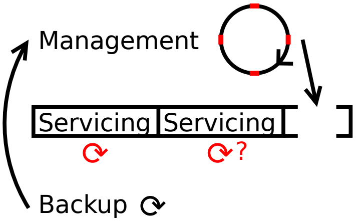
Management

Management Thread

```
if (server->alive == 0) {
    new servicing_thread()
    schedule(starving_thread)
} else {
    await(timeout)
}
```

Backup Thread

```
server->alive = 0
wakeup(management_thread)
```



Scheduling

Policy:

POSIX FIFO

Priorities:

4	Management Thread
3	Servicing Threads
2	Backup Thread

Management

Management Thread

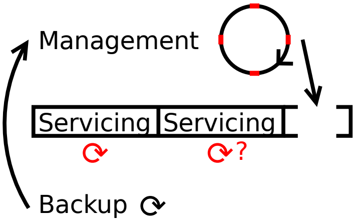
```
if (server->alive == 0) {
    new servicing_thread()
    schedule(starving_thread)
} else {
    await(timeout)
}
```

Backup Thread

```
server->alive = 0
wakeup(management_thread)
```

Timestamp Scheduling

```
schedule(thread)
  0 servicing_thread_1
  0 servicing_thread_2
  0 servicing_thread_3
  0 server
```



Scheduling

Policy:

POSIX FIFO

Priorities:

4	Management Thread
3	Servicing Threads
2	Backup Thread

Management

Management Thread

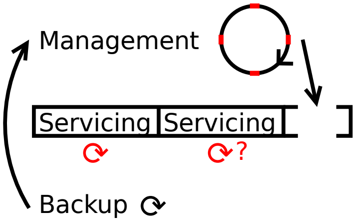
```
if (server->alive == 0) {
    new servicing_thread()
    schedule(starving_thread)
} else {
    await(timeout)
}
```

Backup Thread

```
server->alive = 0
wakeup(management_thread)
```

Timestamp Scheduling

```
schedule(thread)
  0 servicing_thread_1
  0 servicing_thread_2
  0 servicing_thread_3
  1 server
```



Scheduling

Policy:

POSIX FIFO

Priorities:

4	Management Thread
3	Servicing Threads
2	Backup Thread

Management

Management Thread

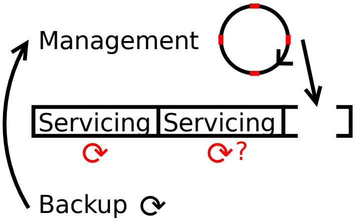
```
if (server->alive == 0) {
    new servicing_thread()
    schedule(starving_thread)
} else {
    await(timeout)
}
```

Backup Thread

```
server->alive = 0
wakeup(management_thread)
```

Timestamp Scheduling

```
schedule(thread)
  1 servicing_thread_1
  0 servicing_thread_2
  0 servicing_thread_3
  1 server
```



Scheduling

Policy:

POSIX FIFO

Priorities:

4	Management Thread
3	Servicing Threads
2	Backup Thread

Management

Management Thread

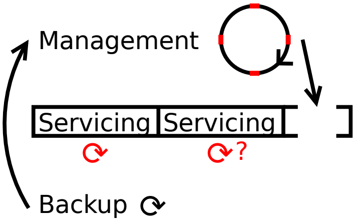
```
if (server->alive == 0) {
    **new** servicing_thread()
    schedule(*starving_thread*)
} else {
    await(timeout)
}
```

Backup Thread

```
server->alive = 0
wakeup(management_thread)
```

Timestamp Scheduling

```
schedule(thread)
  1 servicing_thread_1
  1 servicing_thread_2
  0 servicing_thread_3
  1 server
```



Scheduling

Policy:

POSIX FIFO

Priorities:

4	Management Thread
3	Servicing Threads
2	Backup Thread

Management

Management Thread

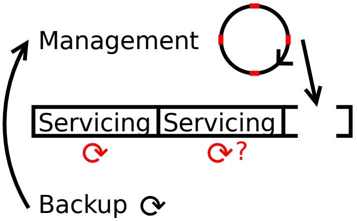
```
if (server->alive == 0) {
    new servicing_thread()
    schedule(starving_thread)
} else {
    await(timeout)
}
```

Backup Thread

```
server->alive = 0
wakeup(management_thread)
```

Timestamp Scheduling

```
schedule(thread)
  1 servicing_thread_1
  1 servicing_thread_2
  1 servicing_thread_3
  1 server
```



Scheduling

Policy:
 POSIX FIFO

Priorities:

4	Management Thread
3	Servicing Threads
2	Backup Thread

Management

Management Thread

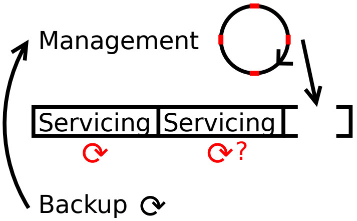
```
if (server->alive == 0) {
    new servicing_thread()
    schedule(starving_thread)
} else {
    await(timeout)
}
```

Backup Thread

```
server->alive = 0
wakeup(management_thread)
```

Timestamp Scheduling

```
schedule(thread)
  1 servicing_thread_1
  1 servicing_thread_2
  1 servicing_thread_3
  2 server
```



Scheduling

Policy:

POSIX FIFO

Priorities:

4	Management Thread
3	Servicing Threads
2	Backup Thread

Management

Management Thread

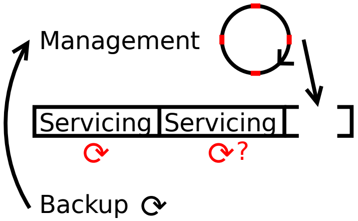
```
if (server->alive == 0) {
    **new** servicing_thread()
    schedule(*starving_thread*)
} else {
    await(timeout)
}
```

Backup Thread

```
server->alive = 0
wakeup(management_thread)
```

Timestamp Scheduling

```
schedule(thread)
  1 servicing_thread_1
  1 servicing_thread_2
  1 servicing_thread_3
  2 server
```



Scheduling

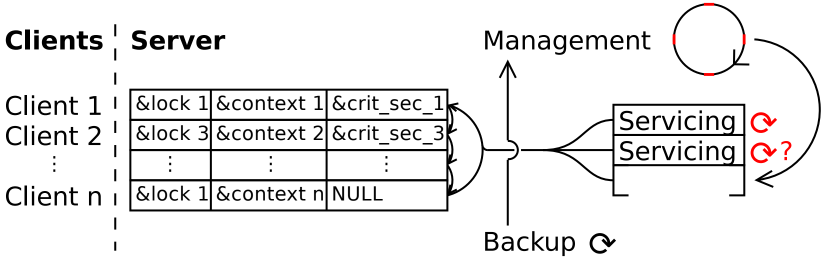
Policy:

POSIX FIFO

Priorities:

4	Management Thread
3	Servicing Threads
2	Backup Thread

Summary


Clients
Server
Client 1
Client 2
⋮
Client n
&lock 1
&context 1
&crit_sec_1
&lock 3
&context 2
&crit_sec_3
&lock 1
&context n
NULL
Management
Servicing
Servicing
?
Backup

RCL Advantages

- improves shared data locality
- scalable
  - only server cache local locking
  - cache synchronisation only between pairs of one client and the server

RCL Advantages

- improves shared data locality
- scalable
  - only server cache local locking
  - cache synchronisation only between pairs of one client and the server

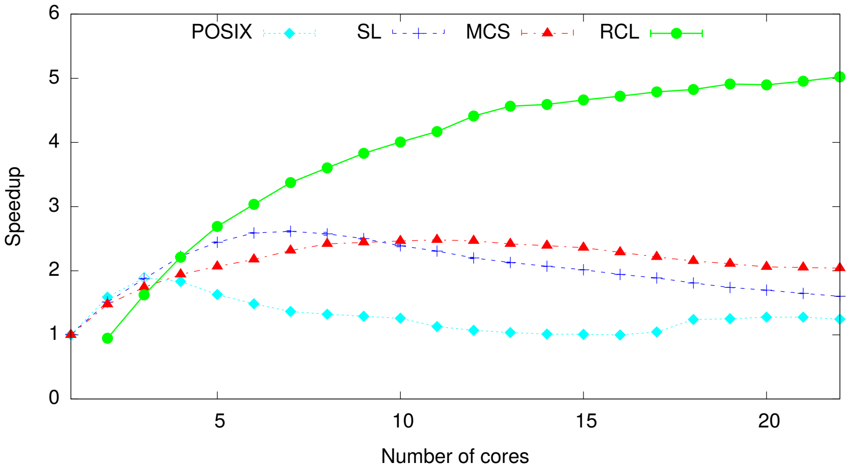
In Future

- allows request combining
- efficiently uses asymmetric many core systems

RCL Disadvantages

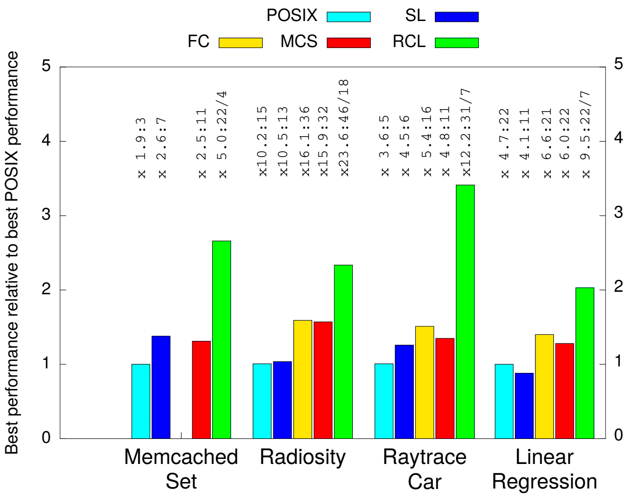
- decreases private data locality
- possibly introduces false serialisation
- hinders fine grained locking
- hardware overhead for server core
- requires root permissions for scheduling in linux
- decreases code readability

Benchmarks



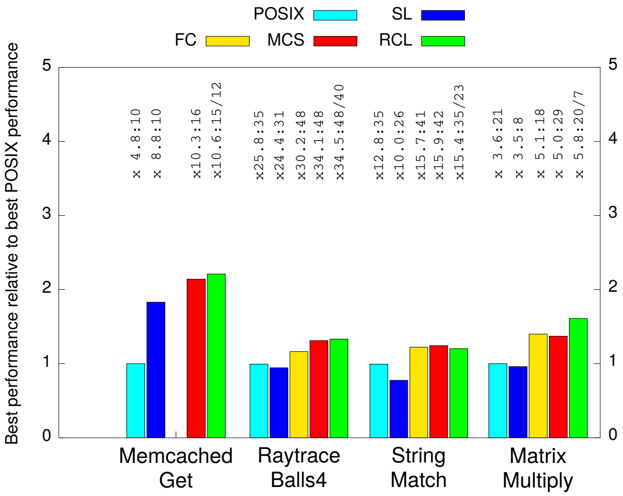
Memcached/Set speedup.

Benchmarks



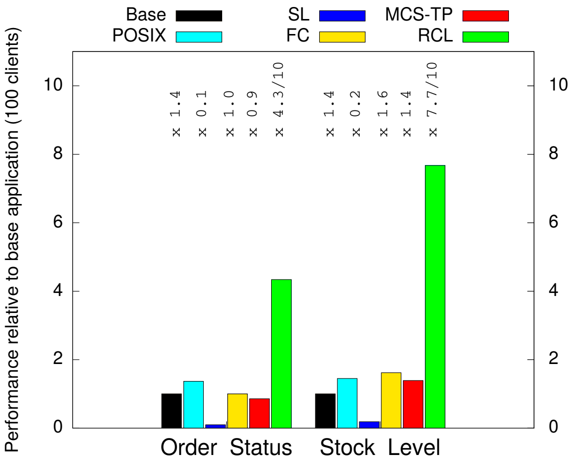
Applications with one lock bottleneck

Benchmarks



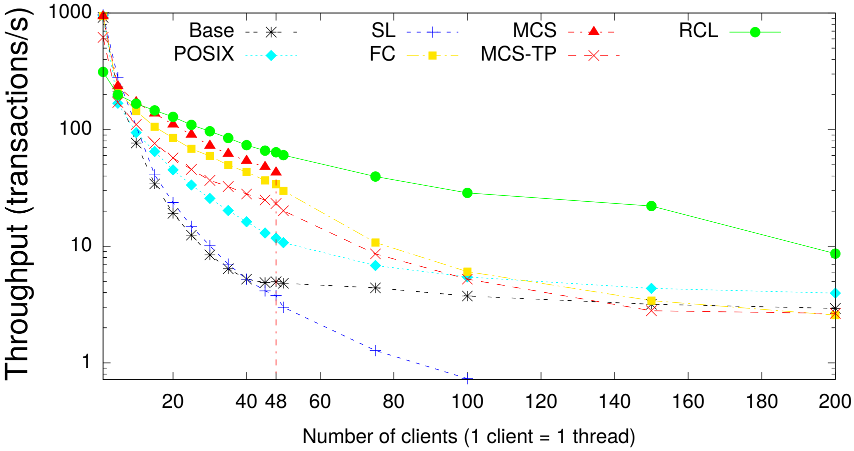
Applications with one lock bottleneck

Benchmarks



Berkeley DB

Benchmarks



Berkeley DB/Stock Level throughput.