王贇 from China LTC

Michael Wang <wangyun@linux.vnet.ibm.com>

Smart Wake-Affine In Scheduler

Agenda

Terminology

Background

Benefit & Issue & Analysis

Figure out the Solution

Our solution in mainline

Thinking

Terminology

wakeup	un-running process need to be waken up to run
waker	The running process which try to wakeup an un-running process
wakee	un-running process to be wakeup
neighbor CPU	Logical cpus which belong to the same core or the same socket
running closely	Processes run on neighbor CPU
related processes	Processes access the same piece of physical memory
pull	Choose a neighbor CPU of waker's to run wakee

Background

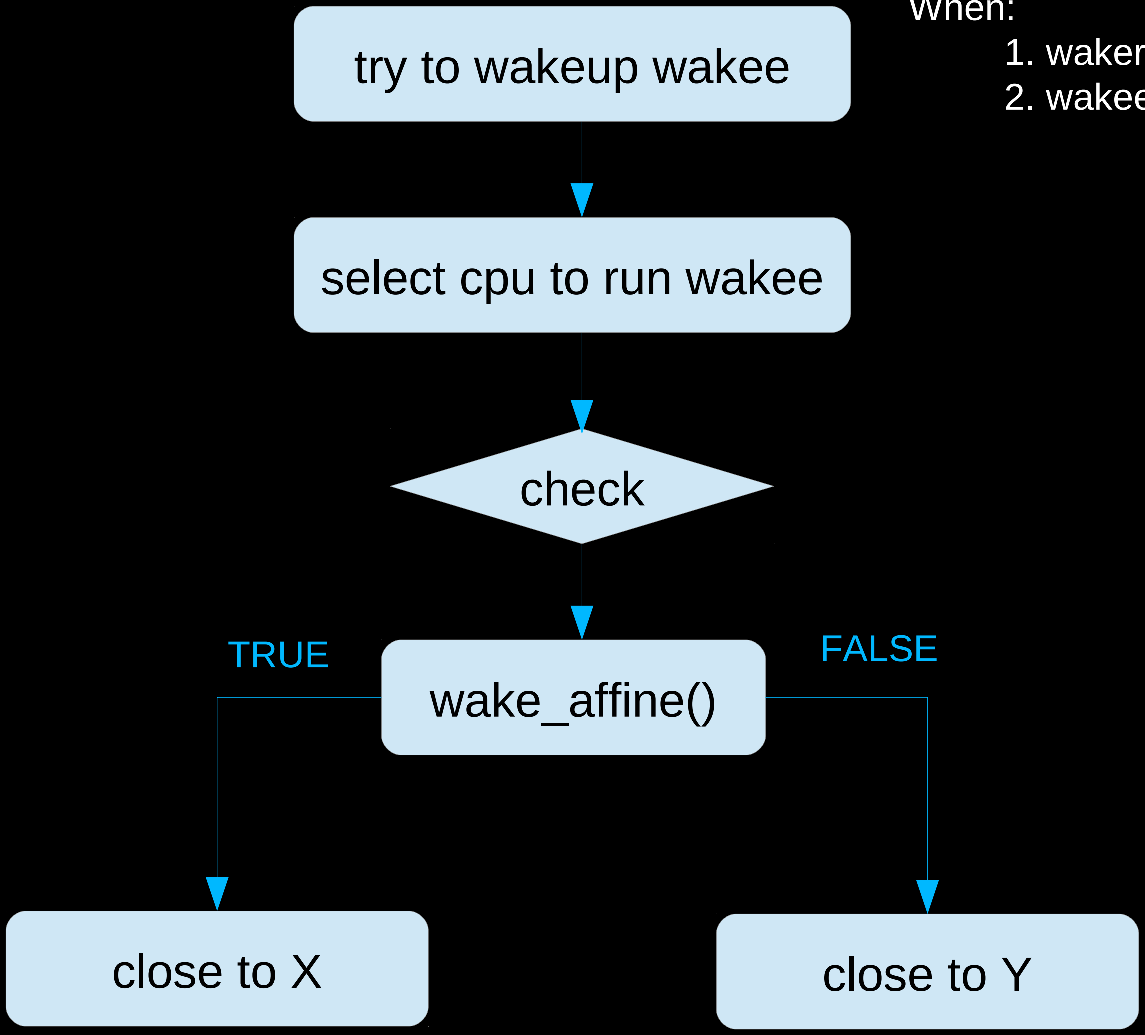
What is wake-affine?

Feature inside scheduler, by which we attempt to make processes running closely

Why wake-affine?

In some cases, related processes running closely will gain benefit, mostly from cache-hit

How wake-affine works?



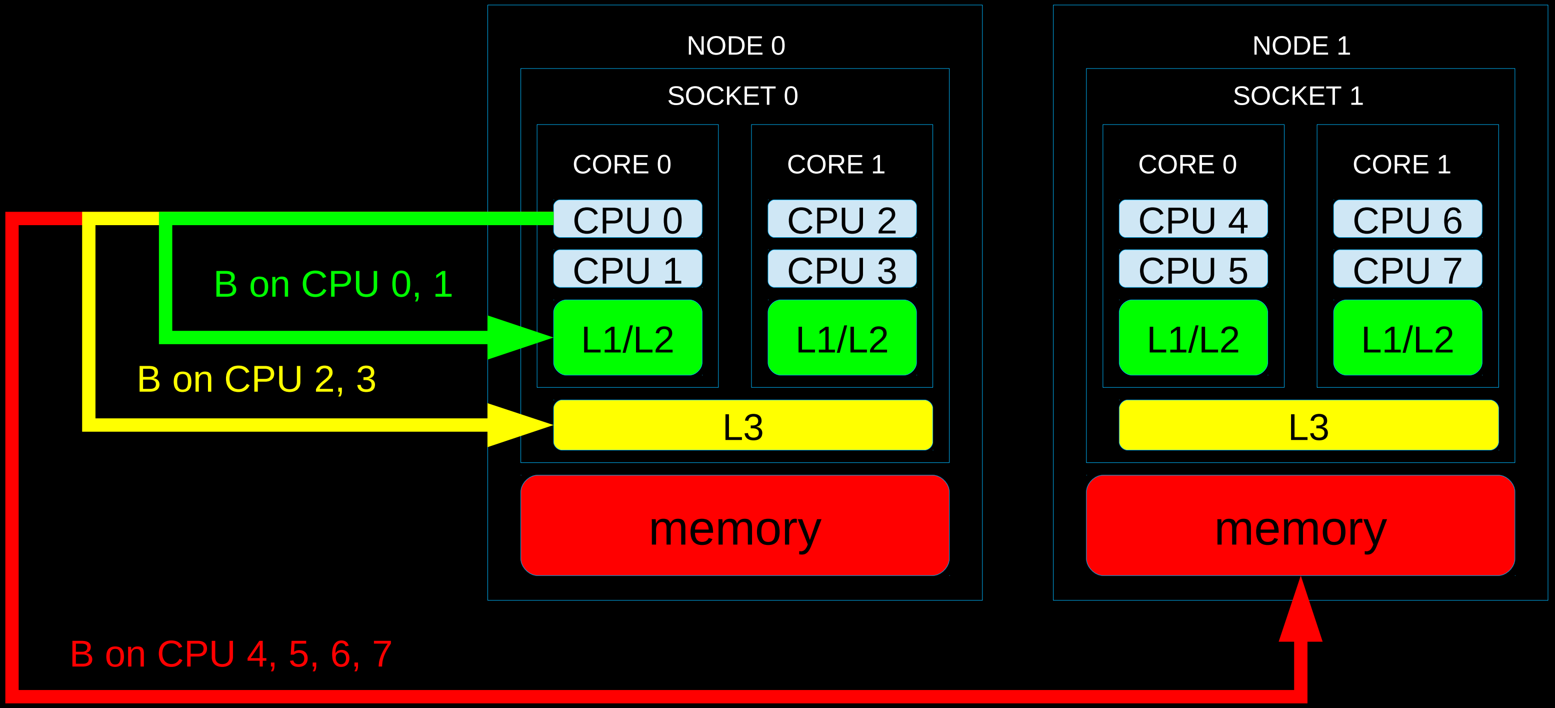
When:

1. waker is currently running on CPU X
2. wakee was last time running on CPU Y

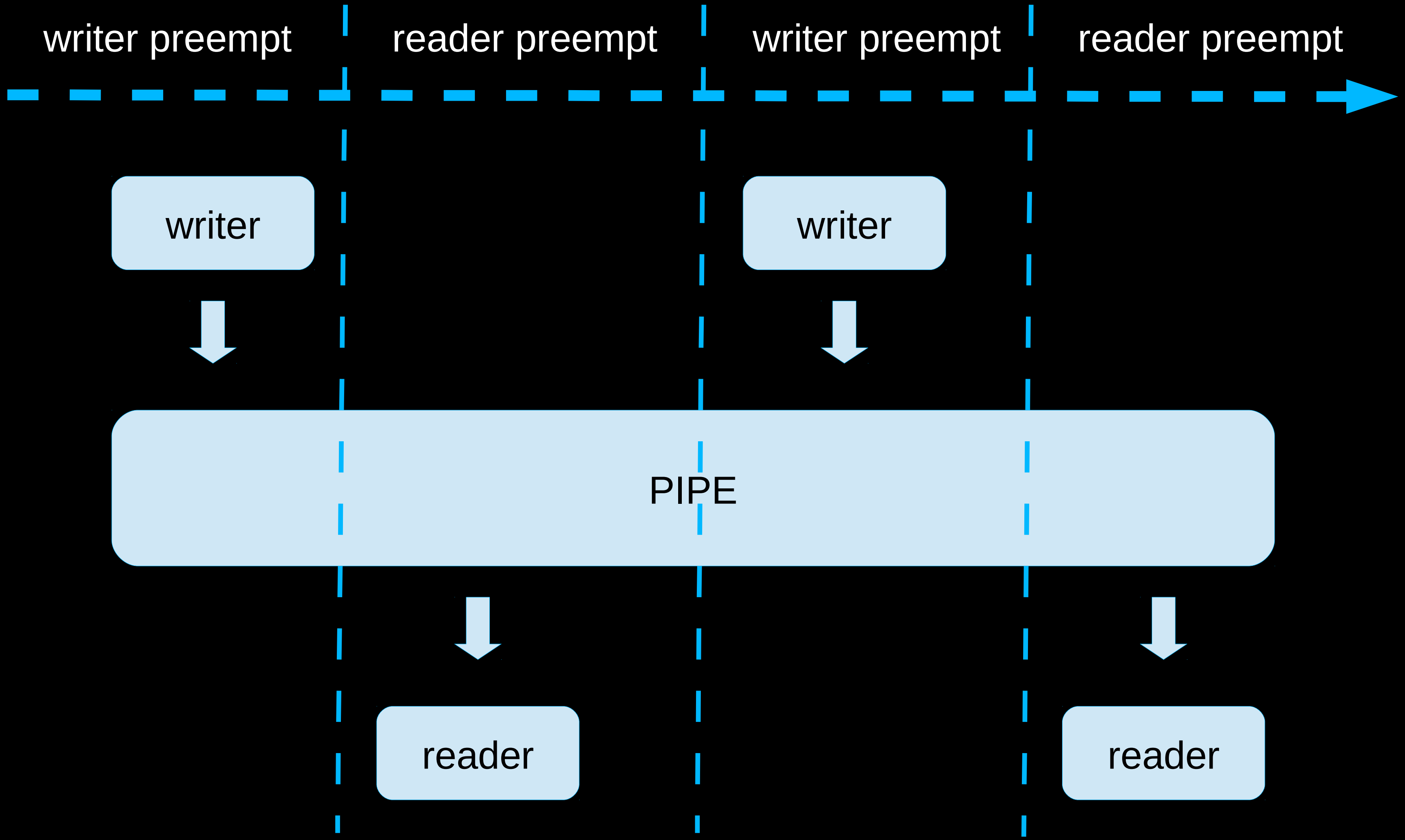
How distance influence the performance?

When:
1. process A is running on CPU 0
2. process B accessed memory of NODE 1
3. the accessed memory has been cached
4. process A want to access the same memory

Then when B run on different CPU...



Special case when writer and reader on same CPU



No wakeup needed

Benefit & Issue & analysis

Benefit

during testing, we found that this feature improved hackbench around 15%

Issue

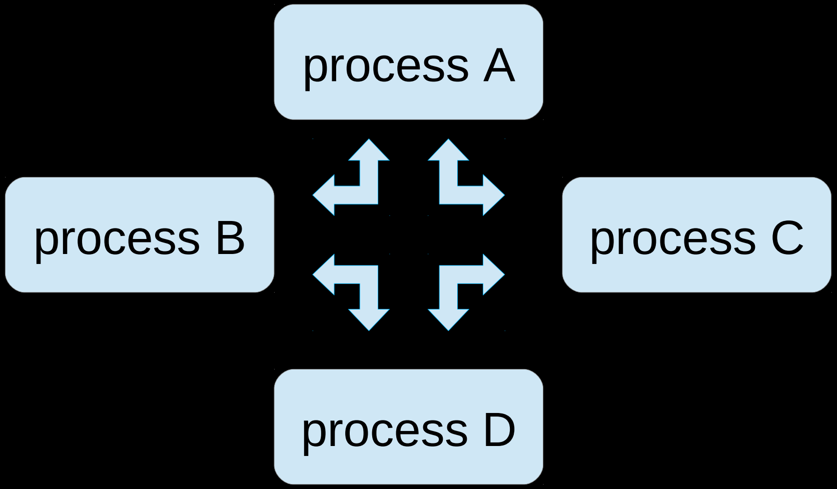
during testing, we found that this feature decrease pgbench around 40%

Analysis

we noticed that pgbench is a benchmark for database, which means the request from clients may finally come into one server, their wakeup relationship are different

wakeup relationship difference

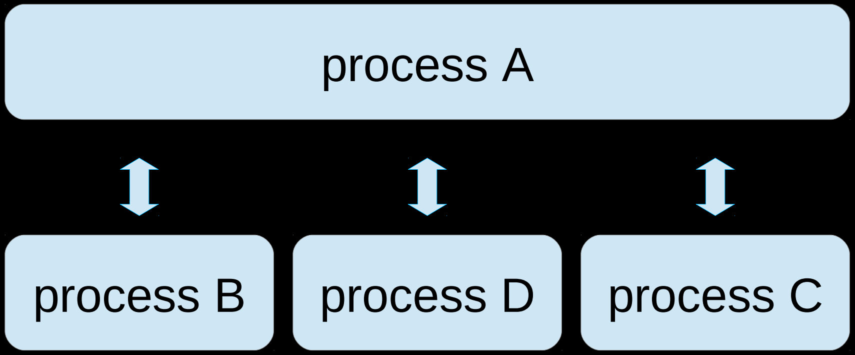
N : N



A, D will wakeup or be waken by B, C.

A pull B, C
B pull A, D
C pull A, D
D pull B, C

1 : N



A will wakeup and be waken by others.

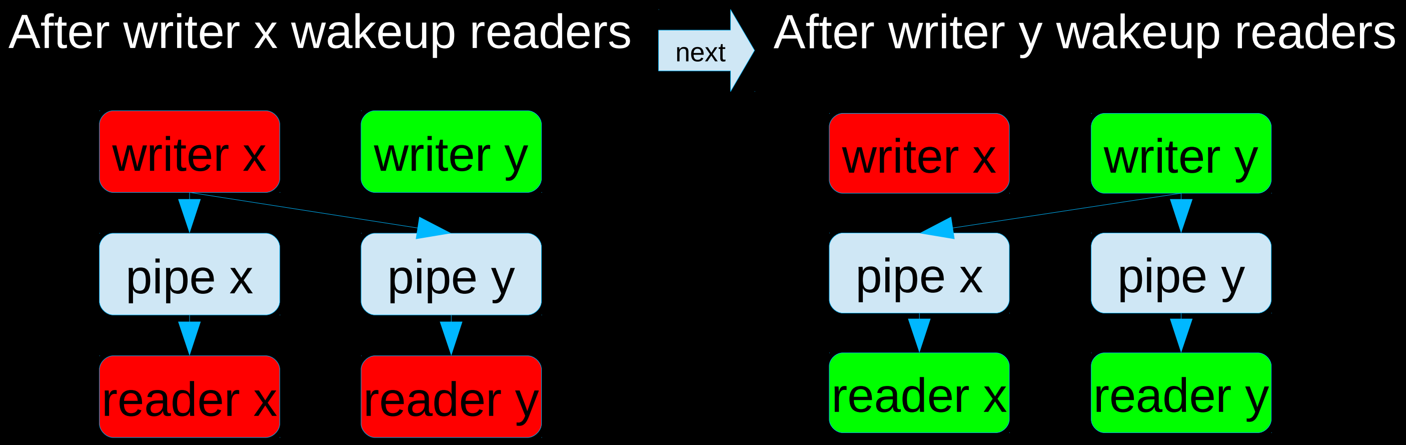
A pull B, C, D
B pull A
C pull A
D pull A

The bad influence of wake-affine

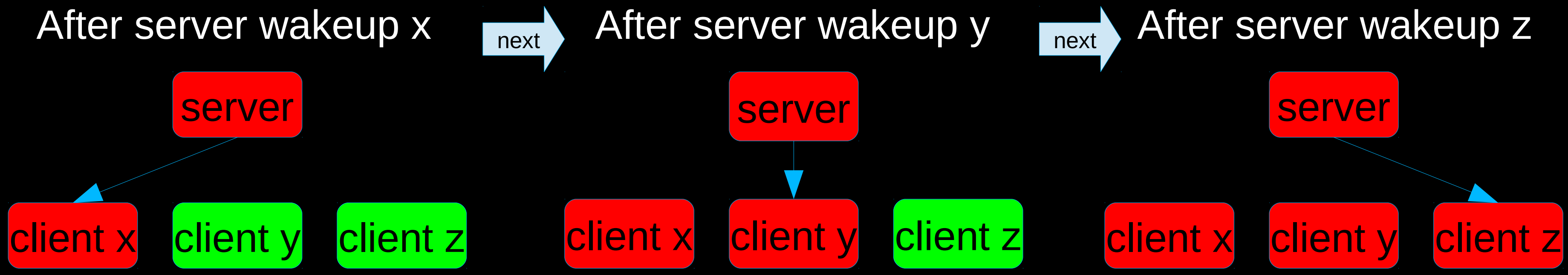
🟥 CPU of NODE 0

🟩 CPU of NODE 1

N : N



1 : N



Analysis

So the issue is that wake-affine has the potential to cause starvation in 1 : N cases, the lost performance caused by:

1. **overhead of wake-affine**
2. **wrong decision of wake-affine which make server starving**
3. **chain reaction caused by server which delay all it's clients**

Figure out the solution

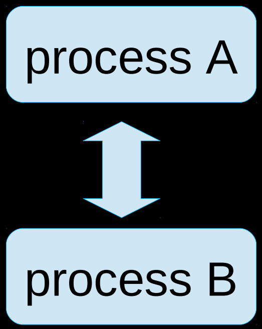
Remove wake-affine logical
- **very easy to implement**
- **hackbench will lost it's benefit**

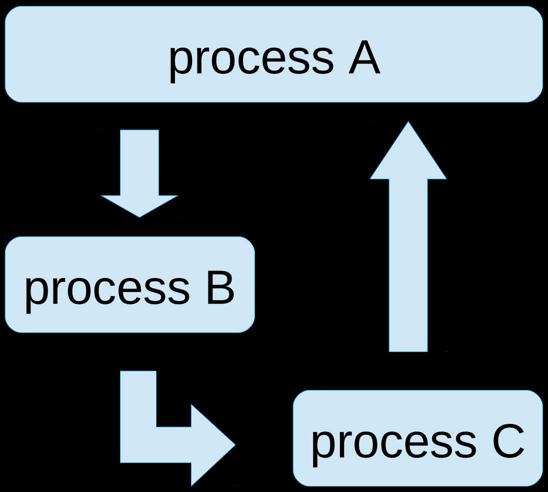
Throttle wake-affine
- **easy to implement**
- **hard for user to use**

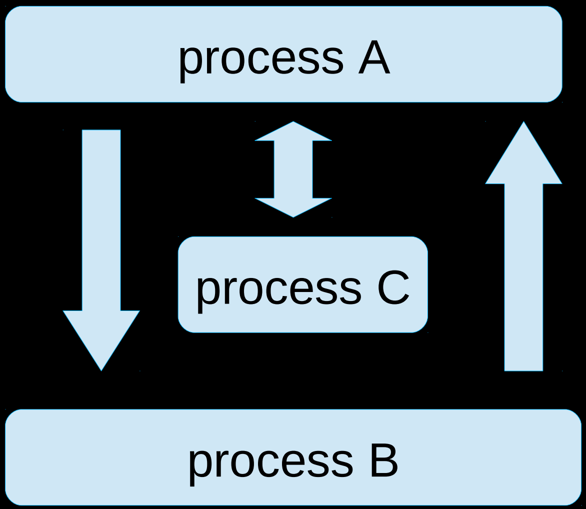
Buddy wake-affine
- **only allow to pull when waker and wakee are keeping wakeup each other.**

- **accurately address the related processes**

Buddy wake-affine







A->wakee = B
B->wakee = A
A->waker = B
B->waker = A

A can pull B
B can pull A

A->wakee = B
B->wakee = C
A->waker = C
B->waker = A

A can't pull B
B can't pull A

A->wakee = C
B->wakee = A
A->waker = B
B->waker = A

A can't pull B
B can't pull A

Buddy wake-affine

Still facing two issues:

1. **overhead is too high.**
2. **lost target in soft-irq context.**

Thus such a solution is required:

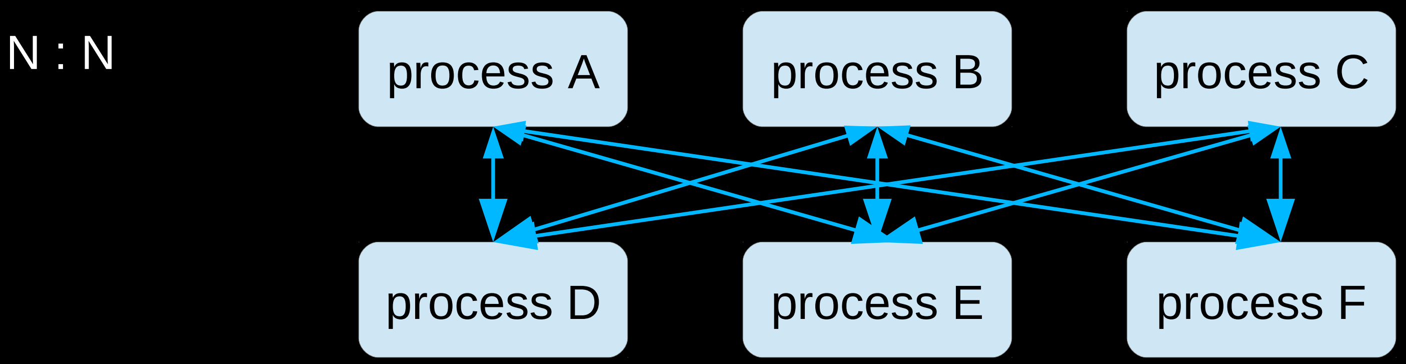
1. **overhead very low.**
2. **automatically detect the bad cases, meanwhile won't kill the good cases.**

Our solution in mainline

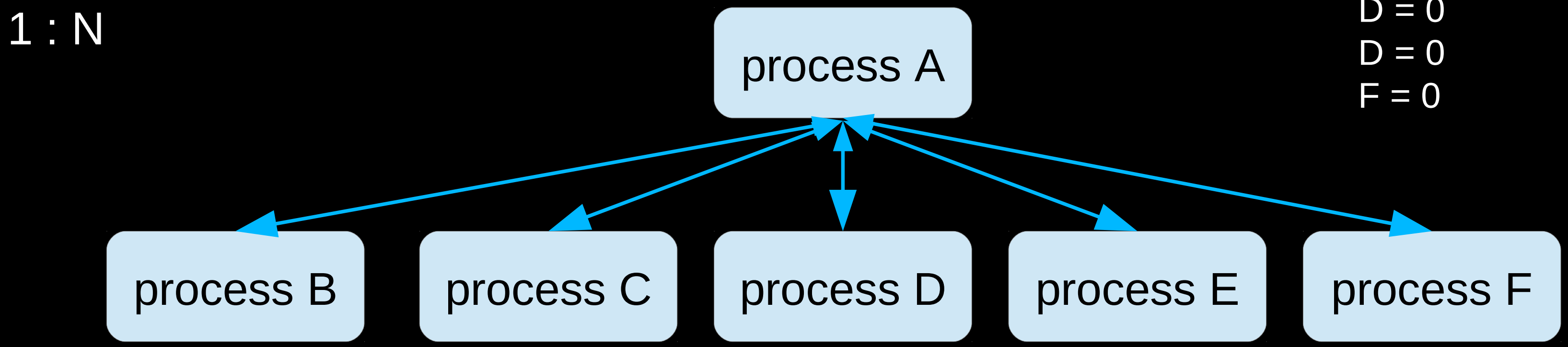
Smart wake-affine

This is a variant of buddy wake-affine solution, recording the flip of wakee switching, stop wake-affine when waker has high flip.

Flip difference



one round flips
A = 2
B = 2
C = 2
D = 2
E = 2
F = 2



one round flips
A = 4
B = 0
C = 0
D = 0
D = 0
F = 0

Our solution in mainline

```
static void record_wakee(struct task_struct *p)
{
        if (jiffies > current->wakee_flip_decay_ts + HZ) {
                current->wakee_flips = 0;
                current->wakee_flip_decay_ts = jiffies;
        }

        if (current->last_wakee != p) {
                current->last_wakee = p;
                current->wakee_flips++;
        }
}
```

```
static int wake_wide(struct task_struct *p)
{
        int factor = this_cpu_read(sd_llc_size);    //top cache-share domain size, added by PeterZ for this solution
        if (p->wakee_flips > factor)
                if (current->wakee_flips > (factor * p->wakee_flips))    //bigger NODE can hold more threads
                        return 1;
        return 0;
}
```

Our solution in mainline

Results:

overhead is low
automatically adjust the scope
get back the lost 40% performance of pgbench
reserve the 15% gotten performance of hackbench
netperf show 3-5% increased performance

Thanks to Fengguang Wu for the data of netperf :)

Thinking

Scheduler used to rely on load-info to make sure it's fairness, but this case show us that only take care the load is not enough in some situation, like this case, process with same load but different wakeup flips should be handled differently.

There are still a wide variety of characteristic we may could use to drive the scheduler decision, by which make it more smart while handling some situation and without conflict with the others.

By analyze the workload of APP, we may also be able to address the key point for optimization, usually the bottleneck and the reason to make it the bottleneck.

Links

commit 7d9ffa8961482232d964173cccba6e14d2d543b2

Author: Peter Zijlstra <peterz@infradead.org>

sched: Micro-optimize the smart wake-affine logic

https://lkml.org/lkml/2013/7/23/866

commit 62470419e993f8d9d93db0effd3af4296ecb79a5

Author: Michael Wang <wangyun@linux.vnet.ibm.com>

sched: Implement smarter wake-affine logic

https://lkml.org/lkml/2013/7/23/875

Thanks to all the folks who involved, special thanks to Mike Galbraith and Peter Zijlstra.

Thanks ;-)

王贇 from China LTC

Michael Wang <wangyun@linux.vnet.ibm.com>