① Job = 1
CPU = 1
run-time = 30
working-set size = 200

-n 1 -L a:30:200

working-set size ⇒ how much cache space it needs to run efficiently

Finished time = 30
CPU utilization = 100
warm = 0.00 ⇒ since default cache size is 100

with cache size 200
-n 1 -M 200 -L a:30:200
We get -
Finished time = 20 ⇒ since warm-rate is 2x, now it runs faster on CPU with warm cache

My experiment not part

this question

CPU utilization = 100

warm = $\frac{10}{20} = \frac{1}{2} = 50.0$

warm_time is 10, hence after 10. Remaining 20, are done in 10 due to warm_rate 2x

(2) -n 1 -L a:30:200 -M 300
Did already above

(3) -n 1 -M 300 -L a:30:200 -T
Reduces 1x then 2x due to warm_rate for next 10 seconds

```
Scheduler central queue: ['a']

  0   a [ 29]
  1   a [ 28]
  2   a [ 27]
  3   a [ 26]
  4   a [ 25]
  5   a [ 24]
  6   a [ 23]
  7   a [ 22]
  8   a [ 21]
  9   a [ 20]
----------------
 10   a [ 18]
 11   a [ 16]
 12   a [ 14]
 13   a [ 12]
 14   a [ 10]
 15   a [  8]
 16   a [  6]
 17   a [  4]
 18   a [  2]
 19   a [  0]
```

-n1 -M 300 -L a:30:200 -r4 -T

Now reduces with 4x

↓

warm_rate

```
Scheduler central queue: ['a']

   0   a [ 29]
   1   a [ 28]
   2   a [ 27]
   3   a [ 26]
   4   a [ 25]
   5   a [ 24]
   6   a [ 23]
   7   a [ 22]
   8   a [ 21]
   9   a [ 20]
-----------------
  10   a [ 16]
  11   a [ 12]
  12   a [  8]
  13   a [  4]
  14   a [  0]
```

④ -n 1 -M 300 -L a:30:200 -C

```
   0   a cache[ ]
   1   a cache[ ]
   2   a cache[ ]
   3   a cache[ ]
   4   a cache[ ]
   5   a cache[ ]
   6   a cache[ ]
   7   a cache[ ]
   8   a cache[ ]
   9   a cache[w]
-------------------
  10   a cache[w]
  11   a cache[w]
  12   a cache[w]
  13   a cache[w]
  14   a cache[w]
  15   a cache[w]
  16   a cache[w]
  17   a cache[w]
  18   a cache[w]
  19   a cache[w]
```

Default warm-time = 10

-n 1 -M 300 -L a:30:200 -w5 -C

↓

warm-time

```
  0   a cache[ ]
  1   a cache[ ]
  2   a cache[ ]
  3   a cache[ ]
  4   a cache[w]
  5   a cache[w]
  6   a cache[w]
  7   a cache[w]
  8   a cache[w]
  9   a cache[w]
--------------------
 10   a cache[w]
 11   a cache[w]
 12   a cache[w]
 13   a cache[w]
 14   a cache[w]
 15   a cache[w]
 16   a cache[w]
 17   a cache[w]
```

`-n 1 -M 300 -L a:30:200 -w 15 -C`

Time ⇒ 15 (to warm #s, 15 remaining caches)

with warm_rate = 2

15 + 15/2 = 15+8 = 23

```
  0   a cache[ ]
  1   a cache[ ]
  2   a cache[ ]
  3   a cache[ ]
  4   a cache[ ]
  5   a cache[ ]
  6   a cache[ ]
  7   a cache[ ]
  8   a cache[ ]
  9   a cache[ ]
--------------------
 10   a cache[ ]
 11   a cache[ ]
 12   a cache[ ]
 13   a cache[ ]
 14   a cache[w]
 15   a cache[w]
 16   a cache[w]
 17   a cache[w]
 18   a cache[w]
 19   a cache[w]
--------------------
 20   a cache[w]
 21   a cache[w]
 22   a cache[w]
```

(5) -n 2 -L a:100:100, b:100:50, c:100:50

↓ CPUs

quantum = 10 3 jobs, cache-size = 100 (default)

Job a ⇒ runtime = 100, working_set_size = 100 (cache space needed to run efficiently)

Job b ⇒ " = 50

Job c ⇒ " = 50

warm_time = 10

Round robin in seq. for multiple CPUs

CPU 1	CPU 2
Job a 0-9	Job b 0-9
cache warmed for Job a	cache warmed for Job b
= 90 Job a	= 90 Job b

Job c	Job a
Job b	Job c
Job a	Job b

No chance of warm cache since cache-size is 100

Other job run overwrites that cache

Total = 300 ⇒ 2 CPUs ⇒ 150 each in parallel

⑥ Time to finally apply cache affinity → (A)

-n 2 -L a:100:100, b:100:50, c:100:50
-A a:0, b:1, c:1

CPU 0 ⇒ Runs a
CPU 1 ⇒ Runs b, c

CPU 0	CPU 1
Job a 0-9	Job b 0-9
cache warmed for Job a	Cache warmed for Job b
Using 100 of cache-size	using 50 of cache-size
= 90 Job a	= 90 Job b
Job a 9-29	Job c 0-9
= 70 Job a	Cache warmed for Job c
	using 50 of cache-size (100)
	= 90 Job c
Job a 29-49	Job b 9-29

= 50 Job a

Job a 49-69
= 30 Job a

Job a 69-89
= 10 Job a

Job a 89-99
= 0 Job a

= 70 Job b

Job c 9-29
= 70 Job c

Job b 29-49
= 50 Job b

Job c 29-49
= 50 Job c

Job b 49-69
= 30 Job b

Job c 49-69
= 30 Job c

Job b 69-89
= 10 Job b

Job c 69-89
= 10 Job c

Job b 89-99
= 0 Job b

Job c 89 99
= 0 Job c

```
Finished time 110

Per-CPU stats
  CPU 0  utilization 50.00 [ warm 40.91 ]
  CPU 1  utilization 100.00 [ warm 81.82 ]
```

Other combinations of 'a', 'b' and 'c' will run slower since b+c uses cache only because 50+50 = cache_size in all other cases with 'a' it will overwritten.

⑦ Super-linear speedup => When you run on N CPUs and get a speedup of more than a factor of N.

-L a:100:100, b:100:100, c:100:100 -M 50

Run of n = 1, 2, 3
↓ CPUs

n = 1, -M 50

Total time = 100 + 100 + 100 = 300

Since cache_size < workingsetsize for all jobs

n = 2, -M 50

Total time = $\frac{100+100+100}{n} = \frac{300}{2} = 150$

n = 3, -M 50

Total time = $\frac{100+100+100}{3} = \frac{300}{3} = 100$

Now let's try with M = 100

n = 1.

Total time = 100 + 100 + 100 = 300

This time cache can be warmed but gets overwritten by next job run

$n = 2$

Total time = $\frac{100 + 100 + 100}{n} = \frac{300}{2} = 150$

Again cache overwritten

$n = 3$

Total time = $\frac{(10 + 45) + (10 + 45) + (10 + 45)}{n}$

$= \frac{165}{3} = \underline{\underline{55}}$

warm = $\frac{45}{55} = 81.82\%$

Each CPU has its own job hence cache becomes effective

Speed up now is $\frac{300}{55} = 5.46\times > N\times$ (N CPUs)

Super-linear speed up

Now I understand the name

I guess since each CPU has a specific job running leading to further speed-up due to cache.

```
Finished time 55

Per-CPU stats
  CPU 0  utilization 100.00 [ warm 81.82 ]
  CPU 1  utilization 100.00 [ warm 81.82 ]
  CPU 2  utilization 100.00 [ warm 81.82 ]
```

⑨ per-CPU scheduling option, -p flag

-L a:100:100, b:100:50, c:100:50

with cache affinity before (⑥)
Total time = 110

with per-CPU scheduling
-P ⇒ how often to peek at other schedule queue.
Default = 30

CPU 0	CPU 1
$Q_0 \rightarrow A \rightarrow C$	$Q_1 \rightarrow B$

1

(10 + 45) = 55

30 (CPU1 not free)

30 + 10 + 10 + 10 = 60

peek (CPU1 free)

60 (CPU1 free, takes "a" job)

70 of c remaining using cache

+35 = 95

95/100

70 of a remaining 10 run "a" using cache

60/2 = 30

60 + 10 + 30 = 100

95/100

```
Finished time 100

Per-CPU stats
  CPU 0  utilization 95.00 [ warm 35.00 ]
  CPU 1  utilization 95.00 [ warm 75.00 ]
```

since free during (55-60)

With -P = 10, 20 no change since no idle job to steal, the other one just ran

with -P = 5,

$Q_0 = a \rightarrow c$, $Q_1 = b$

CPU 0 CPU 1

On five seconds sees Q0 and steals "c" since idle

Stealing happens to and fro but CPU 1, steals after hence always gets the additional job

Once each queue has single job then no stealing since no idle job.

With -P reduced to 5, the time is 90.

```
Finished time 90

Per-CPU stats
  CPU 0  utilization 94.44 [ warm 72.22 ]
  CPU 1  utilization 94.44 [ warm 72.22 ]
```

With -P higher like 90, we get more time

```
Finished time 115

Per-CPU stats
  CPU 0  utilization 95.65 [ warm 26.09 ]
  CPU 1  utilization 78.26 [ warm 60.87 ]
```

With CPUs increase to 3, best time 55 is achieved with 100% CPU utilization and fairness since each job gets a CPU, not idle.

CPUs beyond the number of jobs don't help in perf getting any better, Per-CPU approach especially good for fairness among jobs and balancing among CPUs.

⑨ Played around with various options combinations.