Lab 1: Scheduling

Ankit Goyal
ankit@cs.utexas.edu
CS380L

September 22, 2014

1 Setup

1.1 Hardware

Host Processor: 64 bit 4 core Intel(R) Xeon(R) CPU E3-1270 V2 @ 3.50GHz
Host Memory: 16GB
HyperThreading: Yes
Logical CPUs after Hyperthreading: 8

1.2 Software

Host Operating System: Ubuntu with 3.13.0-34-generic 64 bit kernel.
libcgroup.h used for managing cgroups programatically.
cityhash.h c++ version from Google, used to calculate 128bit Hash values.

2 Single Process Hashing

Single process computed about 722000 Hashes in 5 seconds.

3 Multi Process Hashing

Hyperthreading The processor has 4 physical cores and 8 logical cores (due to Hyperthreading). Table 1 shows that the rate of increase in total time is less when threads are increased from 4 to 8 than when increasing from 8 to 16. Hyperthreading is better than no Hyperthreading but not as better as extra physical cores. Note there are no background processes running.

Number of Hashing Processes	Total Time
4	5.515
8	8.056
16	16.113
32	32.229

Table 1: Showing increase in total time with number of hashing processes.

Hashing Processes #	Background Processes #	Total Time	Throughput	max deviation from mean (fairness)
8	8	15.2418	0.5249	0.968625
8	9	15.5910	0.5131	0.70925

Table 2: Throughput for different number of background and hashing processes.

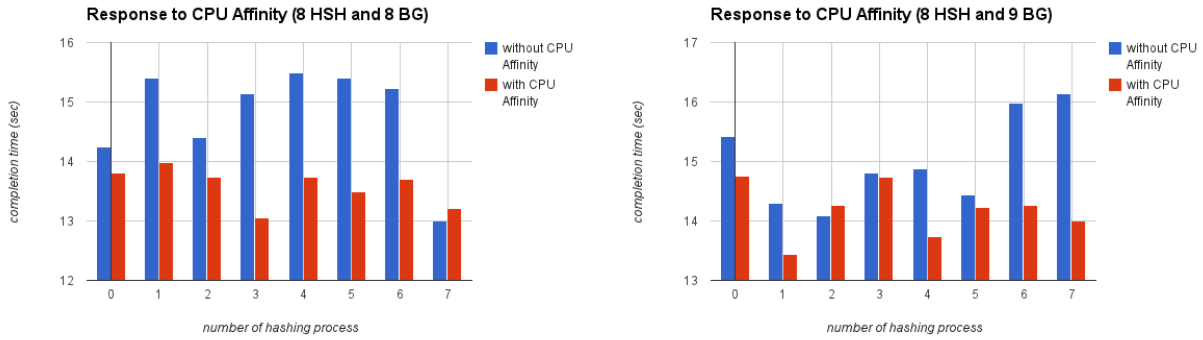
N vs N+1 Processes Table 2b shows the total time taken, throughput and fairness for (8 hashing, 8 background) and (8 hashing, 9 background) processes. Total time is the average value taken over 5 different runs. Throughput in (8 hashing, 8 background) is slightly higher than the throughput in (8 hashing, 9 background) since there are more number of processes to schedule.

4 Throughput

4.1 `sched_setaffinity`

Using `sched_setaffinity`, we attach each hashing process to a specific CPU. CPU Affinity improves cache performance since whenever a processor adds a line to its cache, all other processors needs to invalidate the data in their cache. In certain cases there could be huge improvements in performance due to affinity. The scheduler in 2.5 exhibit excellent natural affinity and prevents bouncing of a process from one CPU to another. CPU affinity could be very effective in case data is shared among different threads (in our case there's no sharing).

Figure 1 shows the effect of `sched_setaffinity` in both (8 Hashing, 8 Background) and (8 Hashing, 9 Background processes). In both cases the throughput increased by 10.9% and 9.5% respectively.



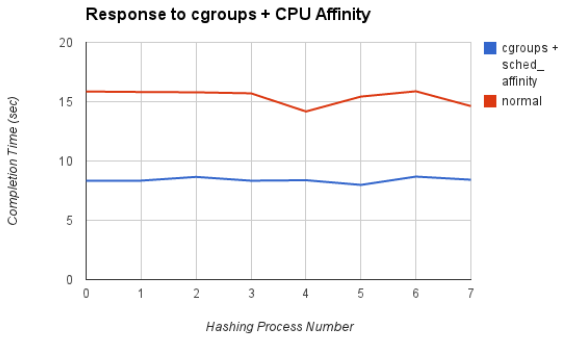
(a) (b)

Figure 1: Response to CPU affinity: `completion time` vs `hashing process number`

4.2 `cgroups`

Control Groups allow you to allocate resources - such as CPU time, system memory, network bandwidth - among user defined group of tasks (processes) running on the system. Using cgroups we can increase the amount of CPU available for hashing processes, which can significantly improve the throughput.
To increase `cpu.share` for hashing processes, a new cgroup for each process was created under the subsystem CPU. The `cpu.share` for all the tasks in these cgroups was increased to 2048 (default value is 1024). As a result all the hashing processes get twice the amount of cpu than other processes with default cgroups (background processes belong to default cgroups).

Figure 2 shows relative time taken by each process with and without cgroups (and sched_affinity). Throughput was increased by 79.5%, since the CPU was given twice to hashing processes than the background processes. It doesn't seem very useful to talk about fairness here, since we deliberately increased the CPU share and tied each hashing process to a different logical CPU.



(a)

Type	Total Time (sec)	Throughput
normal	15.5910	0.5131
cgroup+affinity	8.684	0.9212

(b)

Figure 2: **Response to CPU affinity + cgroups**. (a) Relative time taken by each process with and without cgroup + affinity, (b) Throughput increase of 79.5% for cgroup + affinity

5 Fairness

Time Spent on the lab ≈14 hours

6 References:

1. `http://www.linuxjournal.com/article/6799`
2. `http://www.makelinux.net/books/lkd2/ch10lev1sec5`
3. `http://linux.die.net/man/4/rtc`
4. `http://linux.die.net/man/4/urandom`