Assignment 2

Wenhao Wu

March 20, 2014

In this homework we used two different parallel computing approach to compute the mean, standard deviation, and median of the arrival delay data. In the simulation we use a Dell Inspiron 7420 laptop with Intel(R) Core(TM) i7-3612QM CPU @ 2.10GHz (4 cores 8 threads), 8.00 GB memory, 1TB 5400rpm hard drive, and dual-booted Ubuntu 13.10 OS.

R parallel package method

Our first implementation is based on the R parallel package using the frequency table (histogram) method. Each call of function *funFreqTable()* is in charge of build a frequency table for each file by reading the file from standard input block-by-block. In our bash command, we use *sed* to handle the comma in the quoted field by simply removing it. When *mapply()* returns a list of frequency table, we combine them to get a grand frequency table. In our simulation, we set the number of cores to be from 1 to 6 and the result is shown as follows

	n=1	n=2	n=3	n=4	n=5	n=6
mean	6.98276	6.98276	6.98276	6.98276	6.98276	6.98276
std	30.2205	30.2205	30.2205	30.2205	30.2205	30.2205
median	1	1	1	1	1	1
user(s)	3278.747	3394.827	3437.741	3515.289	4100.097	4740.255
system(s)	112.349	96.335	76.504	62.165	62.311	62.169
elapsed(s)	3142.284	1786.308	1287.308	1056.587	1048.431	1031.973

As shown above, different number of cores does not affect the mean, std and median results as well as the user time. However, the elapsed time is approximately inverse proportional w.r.t the number of cores, when n<=4. However, it seems that when n>4, the elapsed time is saturated, which is in accordance with the fact that our CPU has only 4 cores.

pthread method

Our second implementation is based on a task pool model implemented with pthread. Specifically, we build a structure entity called *data*, whose members include all the filenames of all .csv files, flag value indicating whether each file has been processed by a worker thread or not, and a frequency table for the files processed so far. We start several worker threads, each of which fetches an unprocessed file from *data*, compute its frequency table using fast sample method with sampling probability of 0.0002, and combine the result with the grand frequency table in the *data* entity. Upon accessing the file name and updating the grand frequency table, we need two *mutex* respectively to avoid conflict between different threads. The frequency table is implemented with the *hist* class we defined in Assignment 1. In our simulation, we set the number of worker threads to be from 1 to 6 and the result is shown as follows

	n=1	n=2	n=3	n=4	n=5	n=6
mean	6.34012	6.50635	6.34516	6.07747	6.23435	5.9395

std	32.6222	30.6645	29.9698	30.4578	30.2105	29.2468
median	-1	0	0	-1	-1	0
elapsed(s)	407.265	766.54	571.892	624.263	752.294	969.725

Unforturnately, it appears that the elapsed time is very inconsistent with respect to the number of threads. We use *glances* to monitor the CPU usage and notice that it does not change a lot w.r.t n, so it seems that the thread is not distributed to multiple cores. In the future we will closely examine the control of the threads.