W4112 Database Systems Implementation Project-2 Report

Hsiang-Ho Lin, Rui Lu
Columbia University
Email: {hl2907, rl2784}@columbia.edu

1. Experiment Information

1.1. Which machine were used?

We conducted the experiments on clic machine. The detailed information of the machine, which is obtained by command "*uname -a*" and "*head –n 1 /etc/issue*", is as follows.

uname –a: Linux beijing 3.2.0-77-generic #114-Ubuntu SMP Tue Mar 10 17:26:03 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux

head –n 1 /etc/issue: Ubuntu 12.04.5

1.2. What are the machine's characteristics? (clock frequency, amount of RAM, etc)

The machine has three processors which all have the same type. We used command "*cat /proc/cpuinfo*" to obtain detailed information and their main characteristics are as follows:

processor : 0, 1, 2
vendor_id : GenuineIntel
cpu family : 6
model : 44
model name : Intel(R) Xeon(R) CPU X5650 @ 2.67GHz
cpu MHz : 2666.761
cache size : 12288 KB
fpu : yes
fpu_exception : yes
cpuid level : 11
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts mmx fxsr sse sse2 ss ht syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts nopl xtopology tsc_reliable nonstop_tsc aperfmperf pni ssse3 cx16 sse4_1 sse4_2 popcnt hypervisor lahf_lm ida arat epb dtherm
bogomips : 5333.52
clflush size : 64
cache_alignment : 64
address sizes : 40 bits physical, 48 bits virtual

1.3. Were the machines idle apart from the experiment?

The machine is not idle when we conducted the experiment. We checked this by typing "w" in terminal and found out that there were other users who were using the machine at the same time.

1.4. What complier did you use to compile the code? What compiler options were provided?

We used GCC to compile the code. The options we used are –Wall, -Werror and –msse4.2.

1.5. How did you instrument the code to measure the time taken just for phase 2?

We used two *struct timeval* variables to record the starting time and ending time respectively. Right before calling the function to do probe routine, we recorded the system time into *stv*. Once the probe routine was over, we recorded the system time into *etv*. Finally, we subtracted the value of *stv* from the value of *etv*, and we got the time taken just for phase 2. The code is as following:

```
struct timeval stv;
struct timeval etv;
gettimeofday(&stv, NULL);
/*
 *  probe here
 */
gettimeofday(&etv, NULL)
time = (etv.tv_sec-stv.tv_sec) * 1000000L + (etv.tv_usec - stv.tv_usec);
```

The unit of time is microsecond.

2. Evaluation

2.1. 9-5-9

We used 300 build keys and 30,000 probe keys for tree structure "*9-5-9*". All generated build keys and probe keys are identical for each experiment. We took 20 samples for each comparison.

We implemented a hardcoded mode to probe only for this particular tree structure. So in this section, we first describe the performance differences between binary-search mode and hardcoded mode by comparting the times they take in phase 2. Second, we do the same thing for binary-search mode and non-hardcoded mode. Third, we make a comparison between hardcoded mode and non-hardcoded mode by comparing the above results.

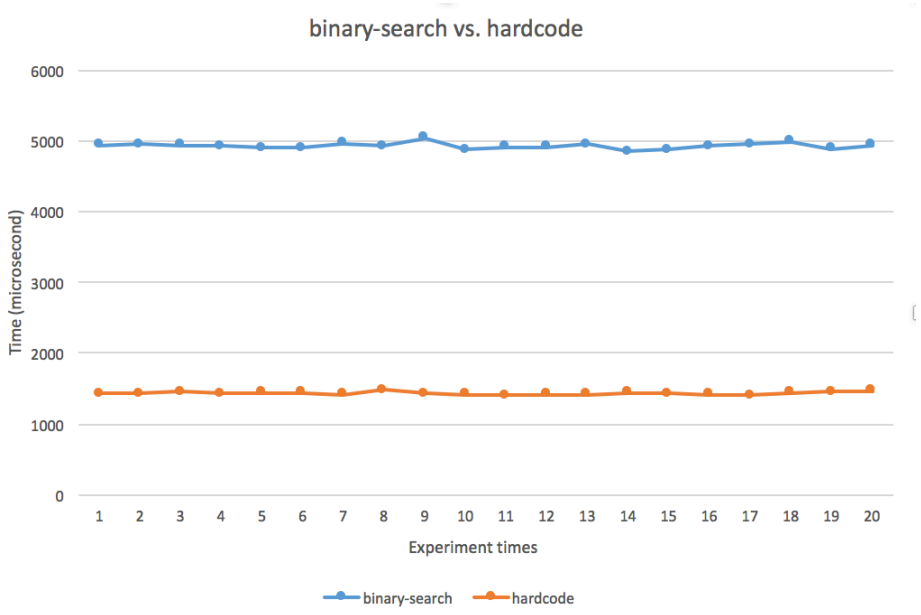

Chart 2.1 binary-search vs. hardcode on 9-5-9

We can know from chart 2.1 that there is a huge difference between the performances of binary-search mode and hardcoded mode. By computing average time, we can get that binary-search mode took 4926.55ms on average to finish the probe phase, while hardcoded mode only took 1432.35ms on average to finish the probe phase. The reasons why there is a huge difference between binary-search mode and hardcoded mode are that hardcoded mode adopts Intel's SSE intrinsic which is fast because of bit operations and hardcoded mode avoids certain overheads such as dereferencing function pointers or checking fanout at each level. Besides, hardcoded mode has no **if** or **while** or **for** loops.

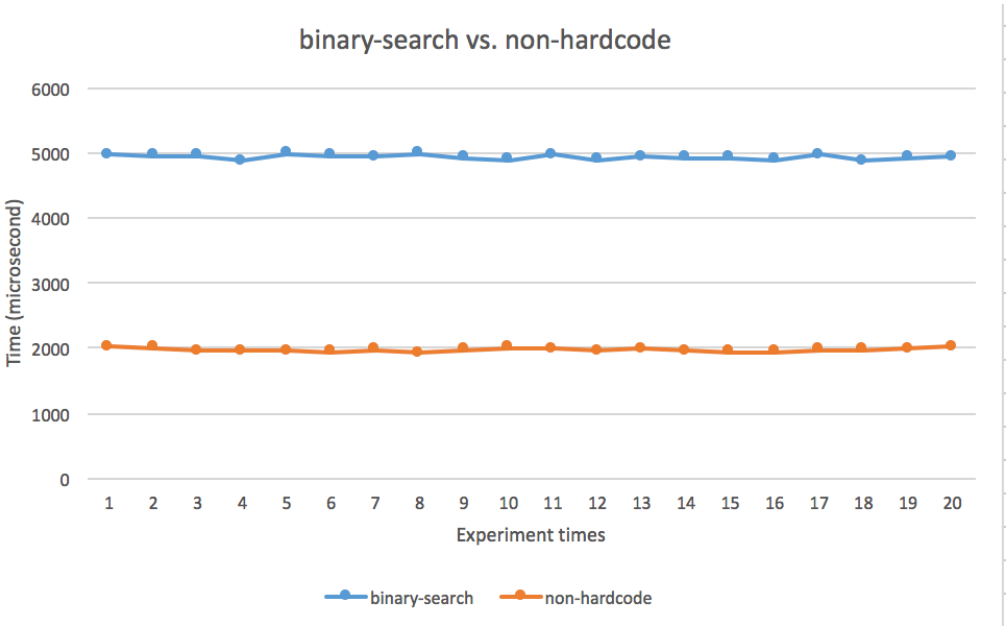

Chart 2.2 binary-search vs. non-hardcode on 9-5-9

We can know from chart 2.2 that there is still a huge difference between the performances of binary-search mode and non-hardcoded mode. By computing average time, we can get that binary-search mode took 4930.6ms on average which is similar with the time that binary-search mode took above. While hardcoded mode only took 1969.4ms on average to finish the probe phase, which is bigger than the time that hardcoded mode took. The reasons why there is a difference between binary-search mode and non-hardcoded mode are the same as above.

However, we also noticed that non-hardcoded mode took much more time than hardcoded mode. The reason we conclude is that the hardcoded mode has no **if** or **while** or **for** statements, so hardcoded mode avoids some certain costs, like checking loop condition.

2.2. 17-17

We used 250 build keys and 30,000 probe keys for tree structure "*17-17*". All generated build keys and probe keys are identical for each experiment. We took 20 samples for this comparison.

According to Chart 2.3, we can easily find out that using SSE instructions to probe is much more efficient than using binary search to probe. The average probing time for SSE instruction is 2045.1ms, however, the average probing time for binary search is 4485.85ms. The reason for that SSE instructions takes less time than binary search when probing the same tree using the same probe keys is the same as the reason described in section 2.1.

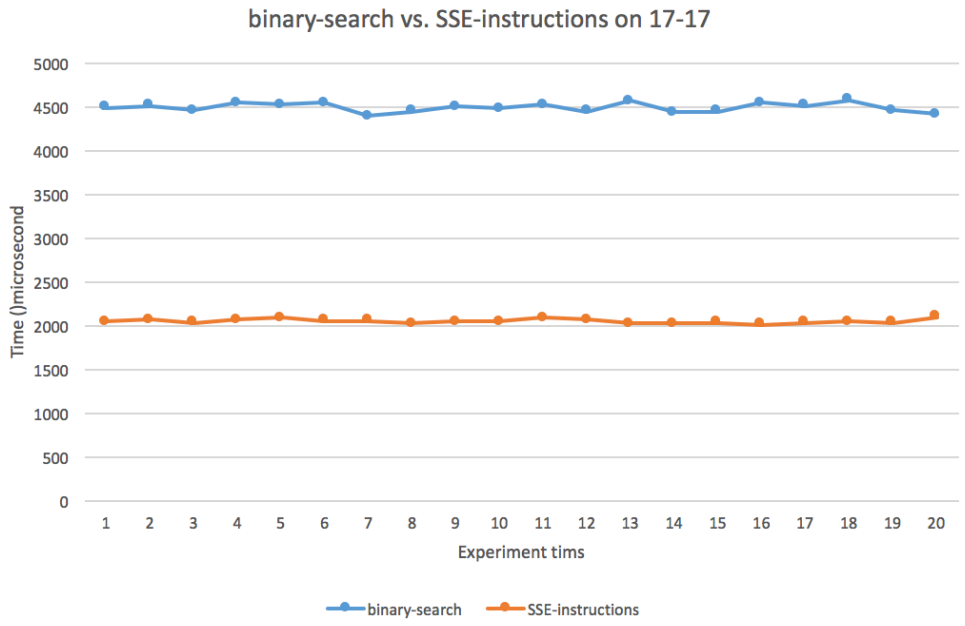

Chart 2.3 binary-search vs. SSE-instructions on 17-17

2.3. 9-5-5-9

We used 1500 build keys and 30,000 probe keys for tree structure "*9-5-5-9*". All generated build keys and probe keys are identical for each experiment. We took 20 samples for this comparison.

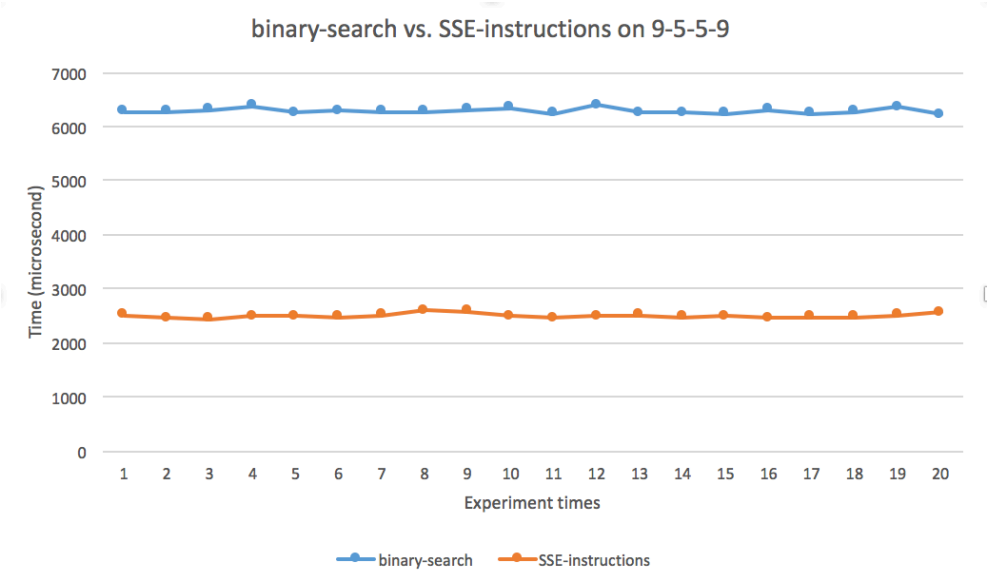

Chart 2.4 binary-search vs. SSE-instructions on 9-5-5-9

According to Chart 2.4, we can easily find out that using SSE instructions to probe is much more efficient than using binary search to probe. The average probing time for SSE instruction is 2501.6ms, however, the average probing time for binary search is 6289.25ms. The reason for that SSE instructions takes less time than binary search when probing the same tree using the same

probe keys is the same as the reason described in section 2.1. Because SSE instructions avoids dereferencing function pointers or checking the fanout at each level. Besides, we also not notice that both times taken by binary-search and SSE-instructions in “9-5-5-9” tree are bigger than those times taken by binary-search and SSE-instructions in both “9-5-9” and “17-17” trees. So we make a conclusion that the more levels or more fanout at each level, the more time taken for probing.