Measurement and Impact of Key File System Attributes
COP4600 Operating Systems, Project 5

By Chengbin Hu and Kimberly Bursum

Abstract:

The overall performance of a computer system depends to a surprisingly large extent on the efficiency of its file system. To understand the extent of the performance impact of various file system attributes it is necessary to devise ways to measure and compare them.

We created a precise time measurement utility program with a 64 bit clock-cycle counter and used it on one of the C4 lab computers to determine that that system has a block size of 4096 bytes, a pre-fetch buffer of 256KB, a file cache of 2MB and 12 direct pointers.

Introduction:

One of the most important aspects of this project was in the development of the empirical tests. We studied closely the attributes of the file system we were testing and developed a timing utility and various test utilities to create test files of the quantity and size needed and read and wrote to them the appropriate number of times to measure the behavior were interesting in seeing.

Methodology:

Platform: The platform we chose to explore was the C4 Lab computer network. This system runs Red Hat Enterprise Linux Server, Release 6.6 (Santiago), Kernel Linux 2.6.32-504.1.3e16.x86-64. GNOME 2.28.2. The file system is EXT4. We isolated one CPU in the network of eight with the setaffinity command and ran all empirical measurements on the same CPU for accurate comparison. The processor is an Intel® Core™ i7-2600 CPU with a 3.40 GHz clock speed.

Timer Accuracy: In this project we needed to measure just a few clock cycles at a time with precision. In previous projects we were able to run a large number of iterations of tests and divide by the iteration number to achieve accurate average results. In this project, running many iterations would have changed the data because we were looking for a very subtle change in rate and it is likely that the operating system would have "optimized" the differences away with many iterations.

We chose to use the highly accurate cycle counter rdtsc which returns a 64 bit cycle count. This counts the number of clock cycles since the machine was last turned on so recording this number twice measures the number of cycles in between with 64 bits accuracy. Given the CPU clock speed (frequency) of 3.40 GHz and knowing that frequency equals number of cycles divided by 1 second, we can calculate that 1 cycle on this CPU takes 2.9411764705 882352941176470588235e-10 seconds.

We validated rdtsc using gettimeofday() and sleep() and found that a program that calls sleep(10) runs about 10 seconds for both gettimeofday() and rdtsc. The values of both gettimeofday() and rdtsc increase linearly with actual time elapsed and therefore are both

reliable for measuring comparative elapsed time.

Measuring disk block size: To measure block size we created a timer utility within our program p5.c. This program takes arguments of what size a file the user wishes and how many times to read it. The program creates a file of the size desired and puts it into a buffer. The function “starts the clock” by calling rdtsc, reads the whole file, then “stops the clock” by calling rdtsc again. The program returns the difference in cycles for that size file which can be used to calculate time difference, thereby giving the time needed to read the file.

We created the program test.c which calls the p5.c program with increasing file sizes to generate comparative data. By charting the difference in time to read a file we noted a sharp step-up that we theorized was a disk block boundary.

We experimented with trying to avoid the prefetching effect to accurately measure sequential reads and theorize that this would require random access reads of a large file on which to do statistical analysis.

Measuring size of pre-fetch buffer: To measure the size of the pre-fetch buffer we used a test.c file that required program p5.c.to run significantly larger files. We made the same calculations as we did in the measurement of disk block size. We calculated time per block from 100 to 1000 blocks and looked for a jump that would signify that the pre-fetch buffer had been exceeded. We calculated the size of the pre-fetch buffer from the difference between block read time and single block read time.

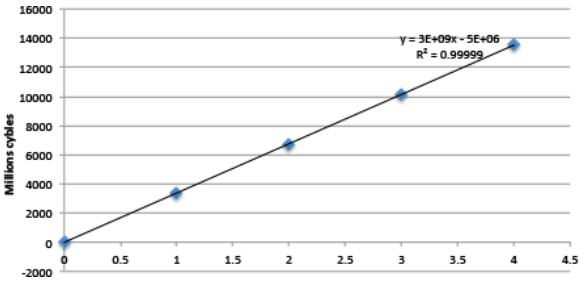
Measuring size of file cache: In our program p5.c we measure the size of the file cache. We write a file that is approximately the size of one file block (because we know that from part one). We read the entire file in and time it and immediately read it again and compare the read time. If the block is in cache then the second read time should be significantly faster than the first because the page is already in memory.

We repeat this measurement for increasing numbers of blocks (the current working set) and cumulatively calculate the time cost for every block. We chart the second time per block and note a sharp jump-up that we theorize is the cache size. When the cache is full, the time per block will increase due to having to go out to disk.

Measuring number of direct pointers in inode: In our program p5.c, we measure several sequential block-sized writes to a new file. We measure the write time and observe that after a certain number of writes there is a sharp increase in write time.

We theorize that when the direct block pointers are used up, indirect block pointers have to be used, leading to the time increase (2 writes versus 1). If each write is block-size then the point immediately before the increase is the number of direct pointers in the inode.

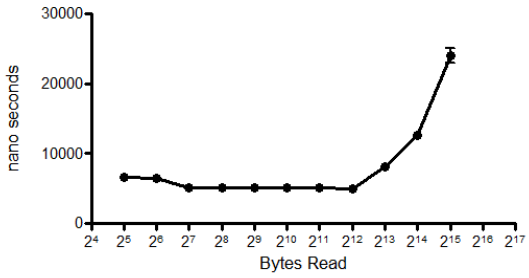
Results:



We compared the rdtsc cycle measurement and gettimeofday() function using sleep() for a known amount of time. The comparison between the results of each shows a good regression with $y = 3E+09x - 5E+06$ $R^2 = 0.99999$ where y is cycles and x is seconds. We can conclude that rdtsc has a linear accuracy for values well over the time we need it to measure.

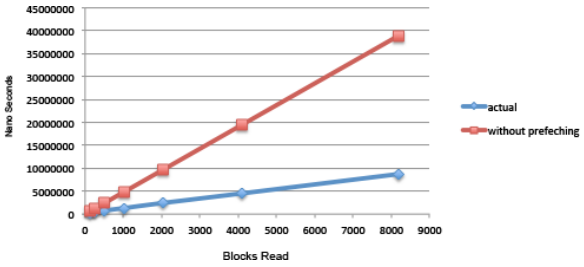
Block size:

We test out p5.c program with increasing file size arguments. We repeatedly measured the time of consecutive reads starting at 32 (2^5) bytes going up to 131,072 (2^17) bytes. Up to 4096 (2^12) bytes the time was steady then at 4096 bytes it increased suddenly.



These increases correspond with a block being full and having to incur the overhead to go to a new block. We can conclude, therefore, that the size of a block is 4096 bytes.

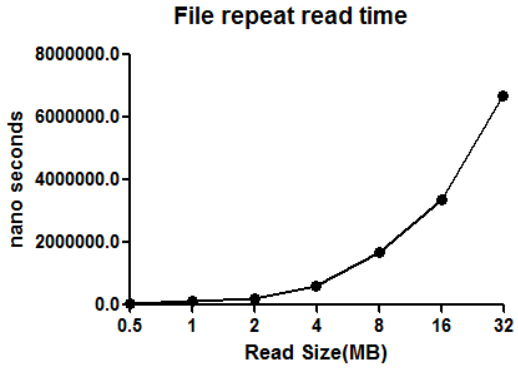
Size of pre-fetch buffer:



We tested our p5.c program with single reads of a large and increasing number of blocks. We observed a significant difference after 1000 blocks which would indicate the pre-fetch buffer was full at that point. The time difference is about 282724.143 in the mean of 3 trials. Therefore the buffer size is about 240KB and we round this to the nearest 2^n value of 60 blocks (or about 256KB).

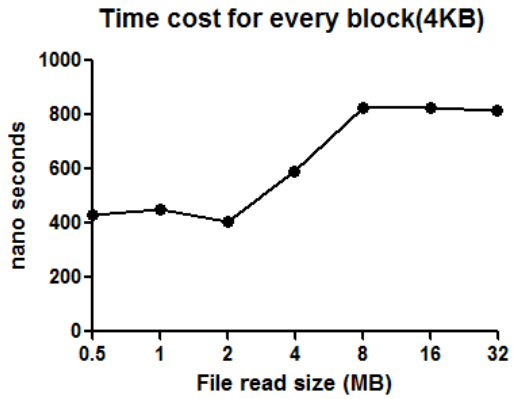
Size of file cache:

As described above, we read a file twice from an increasing number of block-sized files. The raw second read time is shown in the following chart:



To better understand the comparative time we divided the total time by number of blocks and

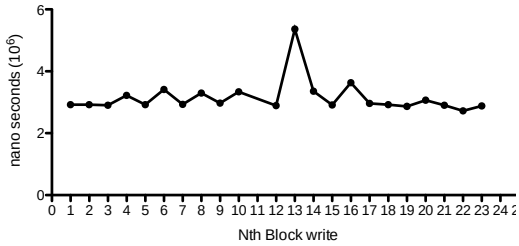
came up with the time per block.



We notice the increase at 2MB which would occur when the cache is full and the program has to go out to disk. The cache size is therefore 2MB.

Number of direct pointers in inode:

As discussed above, we make several sequential block-size writes to a new file. We time the writes and get the following data:



We notice that after the 12th read the read time increases suddenly. If there were 12 direct pointers in that file's inode, when we needed a 13th we would have to use an indirect pointer and this would take one more write causing the increase in time. The direct pointers cause two writes - one to update the inode and one to write the data to disk. With indirect pointers there are three writes: the two above plus we are writing to a new (intermediate) inode. Therefore the number of direct pointers is 12.

Conclusions:

We chose to analyze the C4 lab computers. We used a high precision clock cycle counter to measure the time required for a strategically chosen number of operating system calls on strategically chosen size of file. This revealed that disk block size is 4096 bytes, the pre-fetch buffer size is 256KB, the file cache size is 2MB and that the number of direct pointers per inode is 12.

Teamwork:

We both spent approximately 18 hours each working together and separately on all sections of the project. We both did problem analysis, research and theory of the solution. Chengbin did more of the coding and Kimberly did more of the report writing.