Analysis of Latency Differences between Filesystems

Archit Gupta	Aseem Garg	Ayush Dubey	Shretima Tandon
2008CS10164	2008CS10165	2008CS10168	2008CS10191

April-May, 2011

Introduction

QEMU runs significantly slower on some systems (typically their laptops) and faster on GCL machines. The performance difference is quite big – often Pintos tests timeout on systems that perform poorly. In the following report we provide a possible explanation for following observation. The issue is closely related to the **fsync()** function and its different latencies across different OS's.

1 Approach & Results

Our first observation was that a lot of the tests that were failing had significant filesystem write commands. Tests seemed to do reasonably well when they dealt with filesystem reads and general computation.

We first tried to ascertain that computation is equally fast on the GCL machine(where tests failed : Ubuntu 10.xx) and the palasi server (where tests passed easily). We ran the following code segment on the QEMU emulated pintos which we expected to take the order of a few seconds.

Code I: for testing computation performance:

```
#include <stdio.h>
int main()
{
        int x = 0;
        for(j = 0 ; j <100000; j++){
                for(i=0 ; i<100000; i++){
                        x = x+i+j ;
                }
        }
        printf("dummy variable x is %d", x);
        return 1;
}
```

Result: We found that the code actually ran faster than on the GCL machine.

Hence, we were able to narrow down the problem to being related to file I/O. Next we tried to look at some documentation on the differences between the Ubuntu 9.04 version and the Ubuntu 10.04 version. We found out that a major difference was the shift to the ext4 filesystem on the newer version over the older ext3 in 9.04. We found out from linux documentation on the ext4 filesystem [1] write barriers (barriers concerned with writing to the disk from the on-board disk cache) are on by default on ext4. These barriers significantly increase disk latencies when the

[1] http://kernelnewbies.org/Ext4#head25c0a1275a571f7332fa196d4437c38e79f39f63

higher level application calls the **fsync()** function [2]. This function flushes the hard-drive cache to the hard drive. This can be done in an asynchronous manner when write barriers are off, however, when write barriers are on calls to this function only return when the cache has been successfully flushed. We then tested the following code which writes 2MB worth of data to a file calling fsync() after writing 512 bytes. This code was run on the host OS of different machines (not on Pintos).

Code II: to test the fsync function dependence:

```
#include <time.h>
#include <stdio.h>
#include <fcntl.h>
int main()
{
        clock_t start, end;
        int file, i;
        int  tot = 0;
        long jmp;
        char abc[512];
        time(&start);
        file = open("a.txt",O_RDWR);
        for(i=0; i<512; i++)
        {
                abc[i] = (char)(random()%26 + ((int)'a'));
        }
        for(i=0;i<4096;i++)
        {
                write(file, abc, 512);
                if(fsync(file) != 0)
                        printf("fsync failed\n");
        }
        close(file);
        time(&end);
        printf("%d in %d\n",tot, ((int)(end-start)));
        return 1;
}
```

This code was run on the following machines and gave the performance as shown in Table 2. We were hence, able to deduce the high running time of the GCL machine where the tests were failing. Pintos sets up a **raw** filesystem on QEMU and the QEMU source code for this filesystem [3] calls the function **qemu_fdatasync()** which is responsible for calling the fsync() (or fdatasync() if applicable) upon file writes. Hence, this is the reason for the variable running time for the tests, as implementations of fsync() are different across various filesystems and the new Ubuntu 10.04 suffers from excessive use of fsync() owing to its default enabled write barriers.

2 Solution

Possible solutions may be to ensure that similar disk latencies are obtained across different filesystems. This might be very difficult to achieve by just configuring the disk policy. A possible way to differentiate timeouts in pintos due to buggy code (such as deadlocks) and filesystem latency problems is to enable write-back cache policies for the QEMU process. For this, we can change the

[2] `http://ldn.linuxfoundation.org/article/filesystems-data-preservation-fsync-and-benchmarks-pt-3`
[3] QEMU source file raw-posix.c

Machine	File System	Running Time (sec)
GCL Machine (Ubuntu 10.04)	ext4	84
Palasi	ext4	32
GCL home directory	NFS	9
Personal Laptop (Ubuntu 9.04)	ext3	4
Mac OSX 10.6	Mac OS extended (journaled)	2

Table 1: Comparative Performance of using fsync() function in Code II across different operating systems. Note: these results are not representative benchmarks as the native machines for the operating systems are different.

QEMU configuration from the Pintos perl script such that caching is enabled. We should point out that will drastically decrease the running time of the code and hence will not test whether the assignment has been effectively implemented, however mostly tests will only timeout on the older machines due to a deadlock and doing this can solve this problem. Caching can be enabled by replacing line 652 of pintos perl script (pintos/src/utils/pintos) from -

push (@cmd, '-hda', $disks[0]) if defined $disks[0];

to the new command -

push (@cmd, '-drive','cache=writeback,file='.$disks[0]) if defined $disks[0];

This results in the write-back cache being enabled and we were able to successfully complete our execution of the 'make check' script on the GLC machines where earlier tests were timing out. As an example of the drastic increase in performance the test **'page-merge-stk'** which was earlier timing out (timeout = 60 seconds) executed successfully in less than 3 seconds with write-back cache enabled.