TDT4225 – Assignment 2

Filip F Egge

October 8, 2015

1 I/O Programming

For this part of the exercise i decided to use C as the programming language. I run Ubuntu Linux on my computer and uses the ext4 file system. The block size is 4096. My disk spins at 5400 RPM and has a buffer size of 8192 KB. It has 16383 cylinders, 16 heads, and 63 sectors/track.

Size	Throughput	Time
1 GB	1760 MB/s	581 ms
2 GB	109 MB/s	18788 ms
4 GB	86 MB/s	47503 ms
8 GB	79 MB/s	102818 ms
16 GB	73 MB/s	222426 ms
32 GB	72 MB/s	454004 ms

I was a bit surprised by the results of this test. I did not expect to have such high throughput when testing with 1 GB. I figure it must be buffers and cache that allows for such high speeds when the data is relatively low. When we get to larger than 1 GB it slows down, and normalizes at around 70-80 MB/s. I ran each test 5 times and reported the mean values.

2 Various questions on the file system

Assume data blocks to be 1KByte. What is the largest file size possible in S5FS?

In the S5FS filesystem, the i-node for a file contains a diskmap for all the files data blocks. This disk map contains 13 pointers to disk blocks. The first 10 pointers point to the first 10 data blocks. The next pointer points to a indirect block. An indirect block contains pointers to the next data blocks. Each pointer is 4 byte, to the total number of pointers is 256. Each of theese pointer can point to a data block of 1KB, so the we have 256 KB in addition to the initial 10 KB for the first 10 pointers. The 12th pointer points to a double indirect block. The double indirect block has 256 pointers to indirect blocks, each of them having 256 KB of data. This gives the double indirect block a size of $256 \times 256 KByte = 64MB$. The 13th pointer poinths to a tripple indirect block, containing pointers to 256 double indirect pointers. That gives $256 \times 64MB = 16GB$. If we then sum all the pointers we get $10KB + 256KB + 65MB + 16GB = 17\,247\,250\,432\,Bytes \approx 16GB$.

In S5FS creating or deleting a file requires the I-list to be accessed on disk. Why is this not a big problem?

This is not a big problem because creating or deleting a file does not happend as often as a change to the data blocks. The second reason has to do with crash tolerance. When a system crashes the system needs a complete track of all the i-nodes in use. If changes to i-nodes where cashed then these changes would be

lost if the system where to crash. That would cause severe concequenses for the file system.

How does FFS solve the performance problems of S5FS?

The FFS tries to solve the perfomance problems of S5FS by increasing the block size and redusing seek time. By increasing the block size, FFS introduces a more complex system for dealing with wasted space. FSS deals with this by creating partial blocks, or fragments. To reduce seek time FFS tries to keep data items related to eachother close to one another on the disk. For instance the data blocks of a file should be close together, also a file's inode should be close to it's data blocks. To do this FSS uses the region approach, in which the disk is divived into regions, and blocks related to eachother are placed in the same region. With this approach comes the problem of trying to place to many blocks into the same region. FSS then also needs to try to find items that need to be spread out.

Compare soft updates with journaling. What are the advantages and disadvantages with the two approaches?

Soft updates is a crash recovery method that is based on the consistency-preserving-approach. The consept is that by writing to disk in the same order that you write to cache you preserve consistency. Journaling is another method based on the transactual-approach. Here each transaction, an atomic operation, is recorded in a journal. If there is a crach, this journal is used to either reverse back to the original state, or to redo the operations as written in the journal.

Compare extent-based block allocation with cylinder groups. What are the advantages and disadvantages with the two approaches?

Extent-based treat files as collections of extents, where an extent is a contiguous region of disk space. This approach reduces most of the metadata overhead for large files. Since extents are of variable size this also mean that the disk suffers from fragmentation. This fragementation may therefore end up taking up as much space as the metadata reduction.
Cylinger groups is the consept of dividing the disk into regions in which related blocks are located. Inside each group there is space set aside for inodes, data, and indirect blocks. Directory inodes and content are placed in the same group. Subdirectories are placed in their own groups. This gives short seek times for inodes in the same directory.

Describe briefly the different RAID levels.

Raid 0
Splits data across multiple disks without parity, redunancy, or fault tolerance. If one disk fails the entire disk stack will fail.

Raid 1
This is a data mirroring without parity or striping. Can continue working if one disk fails.

Raid 2
Data striping at the bit level with hamming code for error detection. Synchronized spinning allows for extremely high data transfers. Only original level not currently used.

Raid 3
Byte level striping with a dedicated parity disk. Requires all disks to spin synchronized. Replaced by RAID 5.

Raid 4
Byte level striping with a dedicated parity disk. Rarely used, replaced by RAID 5.

Raid 5
Block level striping with distributed parity. Requires all but one present to operate. Requires at least 3 disks.

Raid 6
Block level striping with two distributed parity blocks.