TDT4225: Exercise 1

Christoffer Tønnessen

Task 1: Write speeds

Size	MB/s	ms
1 GB	613 MB/s	1 671 ms
2 GB	643 MB/s	3 184 ms
4 GB	663 MB/s	6 182 ms
8 GB	675 MB/s	12 135 ms
16 GB	652 MB/s	25 137 ms
32 GB	618 MB/s	53 000 ms

Using a MacBook Pro Retina 15” late 2013 model.
Operating System: Mac OS X 10.10.1 (ten ten ten.1)
Processor: 2,3 GHz Intel Core i7
RAM: 16 GB 1600MHz DDR3
Storage: SSD 500GB, format: Mac OS Extended (Journaled)

My test results seem reasonable, at least in the correct magnitude. A more professional tool was taken from the app store called “Blackmagic Disk Speed Test” and it showed almost the same results:

The difference could be from rust being an unstable language, or my code being suboptimal.

Task 2: Various questions

Assume data blocks of 1KByte. What is the largest file size possible in S5FS?
The file size is directly linked to the amount of blocks one can fit into a file's inode. In a S5FS system a disk map contains 13 pointers to disk blocks. 10 of these go directly to blocks while the three last are a indirect pointer to a block, a double direct pointer to a block and a triple indirect pointer to a block.

The indirect block contains 256 (4-byte pointers) to data blocks, which sums up to 256 KB of data. The double indirect block points to 256 indirect blocks, which totals to 64 MB of data. The triple indirect pointer to a block has 256 pointers to 256 double indirect blocks, which totals to 16 GB data. Since the file size is a singed number of bytes and need to fit inside a 32-bit word which limits the file to 2GB.

In S5FS creating or deleting a file requires the I-list to be accessed on disk. Why is this not a big problem?
The freeing and allocation of inodes happen less often that datablocks, so having them on disk aren't that negative. The supernode has a cached list of all nodes so finding them is also pretty fast. This technique of always having the inodes on disk also makes the system crash tolerant. If the system crashed, the inodes will survive. The data blocks and indirect data blocks will be lost, but not the inodes. All information about the allocated and unallocated inodes preserved in a crash.

How does FFS solve the performance problems of S5FS?
S5FS has small block sizes, while FFS has larger block sizes. A small block size means that locality on the disk can be bad. The files can be all over the place, and if the drive is a HDD (who uses that any more?) the parts of the file can be sparse. If this is the case, that would mean that loading a file could take a long time. This is also the reason defragmenting used to be a thing.

FSS also uses block interleaving. This technique is based on not having all blocks sequentially after each other, but rather a bit away from each other. The reason for this is that when you need to read or write to a block, the disk itself wasn't fast enough to use read, so the disk head had passed the block when it was ready again, this means that it needs to take a whole turn around again before the next read. This technique improves this.

The other big thing is that FFS does is that it uses fragments. Fragments means that one block can have parts of multiple files in it. A block consists of different fragments and multiple files can share a block.

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

Soft updates uses a cache with an ordered set of writes that has happened. When a write happens the cache is updated in memory in the same order issued. If a crash happens the cache is lost and the events in the cache never happened. In the event of a crash the system is up and running again immediately, but the writes issued are lost.

Journaling has a log of everything that has happened written to disk. This means that all events that are issued are written down and kept track of. In the event if a crash the system can back up again and go through the log to see what has and hasn't happened yet. This means that we can get the database back to how it was suppose to be even if the writes issued hasn't happened yet.

Compare extent-based block allocation with cylinder groups. What are the advantages and disadvantages with the two approaches?
Extent-based block allocation uses the idea that when you want to read a file, you want to read files close to it afterwards. So multiple blocks are put together in an extent. This part holds information about how many blocks after the data is. In the best case you can read a lot of data in one read, and have very little metadata for a large data file. It can also use scattered data for its total size. This is also beneficial to an extent. If it gets allocated for instance 5 bytes here and 2.7 bits there (random numbers) it can end up taking more space than regular files.

Cylinder groups are also an idea to extend the size of existing blocks. Each cylinder group has a list of inodes and data blocks. The idea is to have datablocks and metadata in the same cylinder group. This reduces rotation latency and fragmentation. Fragmentation because the directory's content doesn't need to be scattered over the whole disk. This is to contrast from extent-based block which strength is that it can use content from all over the disk.

Describe briefly the different RAID levels.
RAID 0 spits data evently between two or more disks. The data here has no redundancy and simply makes it so that the storage is bigger. The data is distributed evenly between the disks, which causes the illusion of faster access.

RAID 1 has a mirror of the data on one disks to the two or more others. Exactly the same data is present on all the disks.

RAID 2 uses a stripe configuration where a part of the total content is placed on all parts of the disks. It uses Hamming code for error correction. You can get extremely high data transfer rates.

RAID 3 has byte-level striping with a dedicated parity disk. It can not handle multiple requests simultaneously since a block of data will, by definition, be spread on multiple disks.

RAID 4 has block-level striping with a dedicated parity disk. It has good random reads, but writes slowly due to the parity on a single disk.

RAID 5 has block-level has striping but not with the parity bits on the same disk. It requires all drives but one to operate. It requires at least 3 disks.

RAID 6 extends RAID 5 with another parity block. It uses block-level striping with two parity blocks. All these blocks are distributed across all member disks.