Shashi Shivaraju
XID:C88650674

Paper Review 5 - The Design and Implementation of a Log-Structured File System

This paper presents a new technique for disk storage management called a log-structured file system. A log-structured file system writes all modifications to disk sequentially in a log-like structure, thereby speeding up both file writing and crash recovery. The authors of the paper argue that over the last decade CPU speeds have increased dramatically while disk access times have only improved slowly causing more and more applications to become disk-bound. This problem has motivated them to devised this new disk storage management technique.

Log-structured file systems are based on the assumption that files are cached in main memory and that increasing memory sizes will make the caches more and more effective at satisfying read requests. As a result, disk traffic will become dominated by writes. A log-structured file system writes all new information to disk in a sequential structure called the log. This approach increases write performance dramatically by eliminating almost all seeks. The sequential nature of the log also permits much faster crash recovery because it needs to only examine the most recent portion of the log.

Recent file systems also have incorporated a log as an auxiliary structure to speed up writes and crash recovery. However, these systems use the log only for temporary storage; the permanent home for information is in a traditional random-access storage structure on disk.The log contains indexing information so that files can be read back with efficiency comparable to current file systems.

The paper describes the implementation and working of LFS works. The basic structures (ie:inode, indirect blocks) are identical to a UNIX file system. The main difference is that the data is written only to a sequential log file. To support random access, various indexes are written to the log as well. Inodes are not written in fixed position, unlike UNIX. The inode locations are stored in a data structure called an inode map, which is also written to the log.

LFS uses segments which are the smallest logical disk unit, and are comprised of up to 1MB of data. Segments are always written sequentially, but they can be intermingled on the disk with other empty segments. This is done by a process called threading, where new blocks are mixed-in with old blocks. This results in a lot of fragmentation. Additionally, blocks must be cleaned in order to provide good performance. The authors have created a segment cleaner to do this, and it works by reading segments, removing non-live data, and writing back a smaller number of segments.

The paper also discusses Crash recovery which included both checkpoints and roll-forward. Checkpoints are a consistent snapshot of the system at a point in time, and are read by the system after a crash to restore the system to that snapshot. Some data can be lost, so roll-forward is also needed. Here, the data written after a checkpoint can be rolled-forward to the latest possible event.

The authors of the paper have implemented a prototype log structured file system called Sprite LFS,which outperforms the current Unix file systems by an order of magnitude for small file writes while matching or exceeding Unix performance for reads and large writes. The users could not tell the difference between the UNIX FFS and LFS, so they resorted to micro-benchmarks to prove their success. They claim that even when the overhead for cleaning is included, Sprite LFS can use 70% of the disk bandwidth for writing, whereas Unix file systems typically can use only 5-10%.The only case where LFS is worse is for reading a file sequentially after it was written randomly.

Shashi Shivaraju
XID:C88650674

The technique is slightly impended by need to maintain large free areas on disk and added cleaning overhead but the author argue that minimal cleaning overhead can be achieved with a simple policy based on cost and benefit.Thus in conclusion the authors argue that the log-structured file system can use disks an order of magnitude more efficiently than existing file systems.