Aakash Saini(2018CSM1001)

March 11, 2019

- **Observations:**Paper proposes the log-based file system to increase the disk-utilization and improve throughput, responsiveness and write-operations completion time as much as an order of Fast File Sytem(FFS). Due to exponential increment in memory of systems majority of the disk operations are write operations and thus write-latency becomes bottleneck in the system performance. Log-structured File(LFS) system buffers the sequence of changes by write operations and then write all these changes to the disk sequentially in single attempt. Data is written to the segments of fetched inode from disk. IMap is updated for the given inode. To improve the efficiency of write operations, LFS breaks the inode into chunks and writes them in-memory segment. Further, garbage collection is needed at regular intervals for freeing the older inodes. Crash recovery is done using check points and roll-forward mechanism. Check-points are maintained in known location of disk and points to iMap chunks which in turn points to inodes and the segments are written to the disk.
- **Limitations:** Segment-cleaning in the paper is not cleared that how this is scheduled. Proposed cleaning mechanisms may ad the complexity to the system as they are need to be allowed directly by OS to "mount" LFS for write-access. For LFS, inode and blocks of a file are not sequential on the disc and thus accessing to them may become slower than FFS.
- **Conclusions:**LFS improves the I/O bandwidth utilization and also the write-call performance by collecting large amount of write-data into the buffer and then writing it to disk in single large I/O operation. Sprite LFS achieves the low-cleaning overheads by using a simple strategy on cost and benefit. Proposed work is applicable for the large file-file accessing also. LFS achieves more better performance then the existing file systems.
- **Future work:**In future, triggers for cleaning can be optimized as present approach slowing the system response time. Write cost function can be implemented in future to update the inode pointers as it can make read of small files more significant. Present works doesn't consider boot time and thus can be considered in future work.