1. Ceph: A Scalable, High-Performance Distributed File System

This paper introduces Ceph, a distributed file system that provides excellent performance, reliability, and scalability. Traditional client/server file systems like NFS, have suffered from scalability problems due to their inherent centralization. In order to improve performance, modern file systems have taken more decentralized approaches. In these traditional systems face scalability limitations due to little or no distribution of the metadata workload.

Ceph decouples data and metadata operations completely by eliminating file allocation tables and replacing them with generating functions (called CRUSH). This allows Ceph to leverage the intelligence present in OSDs and delegate to OSDs not only data access but also update serialization, replication, failure detection, and recovery tasks as well. In addition, Ceph employ an adaptive distributed metadata cluster architecture to vastly improve the scalability of metadata access.

Ceph is currently the hottest SDS (Software Defined Storage) technology in the storage industry. Ceph provides high performance and scalability without SPOF (Single Point of Failure) through a distributed storage system. Because it is SDS, it is not closely related to hardware, so it can run on commodity hardware and can expand sufficiently. Ceph is basically Object Storage that has a flat structure with no hierarchical structure like a file-directory, and accesses objects through key values. Because there is no hierarchy, so there is no physical path, so it doesn't have to manage directory management for a long time. Also, it can be provided as block and file storage by stacking several layers. So, these are strength point of this paper.

But, although Ceph dynamically replicates metadata when flash crowds access single directories or files, the same is not yet true of file data. It is a weak point of Ceph.

Finally, I hope that the specifications and environment settings of the equipment used in the experiment are written in detail.

2. Barrier-Enabled IO Stack for Flash Storage

This paper introduces BarrierFS, file system using the barrier-enabled I/O stack, eliminating the overhead required for guaranteeing the storage order in the modern I/O stack. The existing file system adopts transfer-and-flush as a manner to guarantee the order of writes. However, transfer-and-flush has a limitation that should guarantee the order of write and durability at the same time. Therefore, there is a disadvantage of waiting for the data to be completely durable to the storage when using the ordering guarantee mechanism. To address this problem, BarrierFS provides an ordering guarantee mechanism that complements these disadvantages. By removing the durability from the existing ordering guarantee mechanisms, the next write request can be performed immediately without having to wait for the data to become durable.

BarrierFS can guarantee the order through the barrier write. The barrier write transmits a write

command with a special barrier flag to the storage. When the storage controller receives the barrier flag, it ensures that the previous barrier request is written before the current barrier request or the next barrier request is written after the current barrier request. Therefore, order of write can be guaranteed.

BarrierFS provides dual-mode journaling. In the existing journaling, JBD thread guarantees the order of write by transferring and flushing the journal descriptor block and the journal commit block sequentially. However, the dual-mode journaling has divided the JBD thread into commit and flush threads. The commit thread dispatches the journal descriptor block and the journal commit block with the barrier flag attached, guaranteeing the order. The flush thread flushes the journal descriptor block and the journal commit block to be durable. When an application writes data and triggers a commit thread, the commit thread dispatches a journal descriptor block and a journal commit block and returns. And then, the application can perform the next request immediately. It seems to be the strength that dual mode journaling can avoid blocking of application, it can ensure ordering at the same time.

But BarrierFS is only available in a few eMMC products but unavailable in the standard block device interfaces of commercial SATA, SAS, and NVMe SSDs. It needs device's modification. Plus, when applications need the fsync() semantic, operations occur synchronously with regard to I/Os. I think these can be a weak point.