Distributed Transaction and Self-healing System of DAOS

Zhen Liang[1](✉), Yong Fan[1], Di Wang[2], and Johann Lombardi[3]

[1] Intel China Ltd. GTC, No. 36 3rd Ring Road, Beijing, China
{liang.zhen,fan.yong}@intel.com
[2] Intel Corporation, Santa Clara, CA, USA
di.wang@intel.com
[3] Intel Corporation SAS, 2 Rue de Paris, 92196 Meudon Cedex, France
Johann.lombardi@intel.com

Abstract. The Distributed Asynchronous Object Storage (DAOS) is an open source scale-out storage system designed from the ground up to support Storage Class Memory (SCM) and NVMe storage in user space. DAOS uses an optimized two-phase commit protocol to guarantee atomicity of distributed I/O. This protocol is tightly coupled with the self-healing system of DAOS, in contrast with traditional two-phase commit protocol that is blocking when coordinator fails, this protocol can proceed in presence of failure, and it also has shorter transaction response time than the traditional protocol, these characteristics are important for massively distributed and low latency storage system like DAOS. This paper introduces the distributed transaction and self-healing system of DAOS, and presents the performance benefits of the transaction protocol.

Keywords: DAOS · Distributed storage system · Distributed transaction · Two-phase commit · SCM · Self-healing · Data recovery · Rebuild

1 DAOS Introduction

Distributed Asynchronous Object Storage (DAOS) [1] is a complete I/O architecture that aggregates Storage Class Memory(SCM) and Non-Volatile Memory Express (NVMe) storage distributed across the fabric into globally accessible object address spaces, providing consistency, availability, and resiliency guarantees without compromising performance. It presents a key-value storage interface and provides features such as transactional non-blocking I/O, a versioned data model, and global snapshots.

In order to unleash the full potential of new hardware technologies, the new stack provides byte-granular shared-nothing interface, it can support massively distributed storage for which failure will be the norm while preserving low latency and high bandwidth access over the fabric.

J. Nichols et al. (Eds.): SMC 2020, CCIS 1315, pp. 334–348, 2020.
https://doi.org/10.1007/978-3-030-63393-6_22

1.1 DAOS System Architecture

DAOS takes advantage of next generation technologies like SCM and NVMe. It bypasses all of the Linux kernel I/O, runs end-to-end in user space, and avoids system calls during I/O.

As shown in Fig. 1, DAOS is built over three building blocks. The first one is persistent memory and the Persistent Memory Development Toolkit (PMDK) [14]. DAOS uses it to store all internal metadata, application or middleware key index, and latency sensitive small I/O. DAOS uses a hybrid approach to optimize the trade-offs between cost, performance, and capacity, this requires the second building block, NVMe SSDs and the Storage Performance Development Kit (SPDK) [13] software, to support large streaming I/O. The DAOS service can submit multiple I/O requests via SPDK queue pairs in an asynchronous manner from user space, and create persistent memory indexes for data in SSDs. Libfabric [12] and an underlying high performance fabric is the third build block for DAOS. It is a library that defines the user space API of OFI, and exports fabric communication services to application or storage services. The transport layer of DAOS is built on top of Mercury [11] with a Libfabric/OFI plugin.

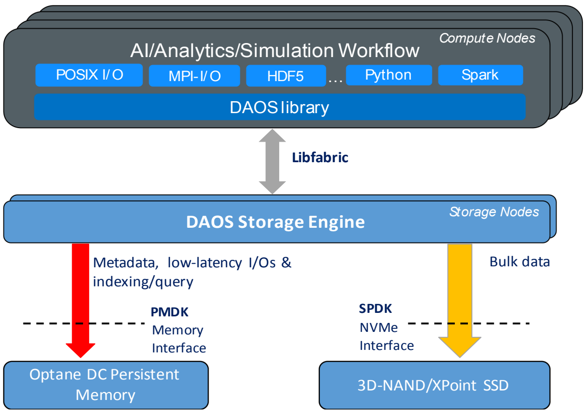


Fig. 1. DAOS architecture

1.2 Data Protection and Distributed I/O

In order to prevent data loss, DAOS provides both replication and erasure coding for data protection and recovery. When data protection is enabled, DAOS objects are stored across multiple storage nodes for resilience. If a failure happens on a storage device or server, DAOS objects are still accessible in degraded mode, and data redundancy is recoverable from replicas or parities.

DAOS distributed I/O for data protection is a primary-slave model: The primary server forwards client requests to slave servers. This model is slightly different from a traditional one. As shown in Fig. 2, the primary server forwards the RPC and RDMA descriptor to slave servers. All servers will then initiate an RDMA request and get the data directly from the client buffer. DAOS chooses this model because the fabric bandwidth between client and server is much higher than the bandwidth between servers in most HPC environments.

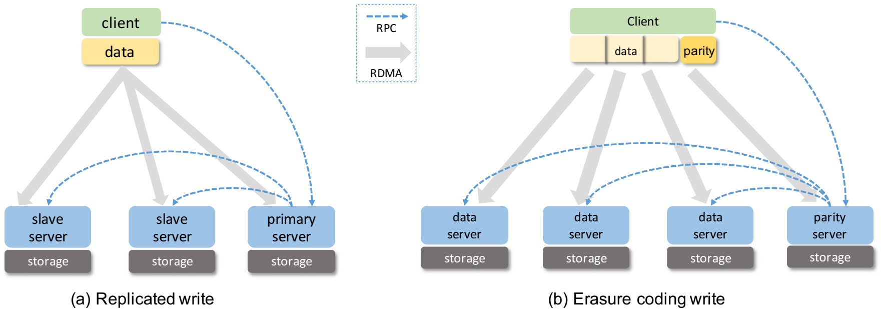


Fig. 2. DAOS distributed I/O

DAOS uses an optimized two-phase commit protocol, which is tightly coupled with self-healing system, to ensure atomicity of the distributed I/O for data protection. The main focus of this paper is introducing how this protocol overcomes the blocking problem of two-phase commit, supports low transaction response time and reduces the number of messages between servers as presumed commit protocol [5].

1.3 Algorithmic Object Placement and Redundancy Group

DAOS storage is exposed as objects that allow user access through a key-value or key-array API. In order to avoid scaling problems and the overhead of maintaining per- object layout metadata, a DAOS object is only identified by an ID that has a few encoded bits to describe data distribution and the protection strategy (replication or erasure code, stripe count, etc.). DAOS passes object ID and storage pool membership to a pseudo-random based placement algorithm to compute object layout, this process is called algorithmic object placement [4].

Layout of a distributed object can consist of N redundancy groups, each redundantly storing a subset of object data. For replication, each member of a redundancy group stores one replica of the same object shard, whereas for erasure coding, a redundancy group is equivalent to a parity group. The distributed transaction described in this paper only applies to I/O against one redundancy group, thus a redundancy group is the equivalent of transaction group within context of this paper.

1.4 Self-healing System

In a distributed storage system, rectification of system faults is important because Mean Time Between Failures(MTBF) of the system decreases when the system scales to more storage nodes, if the storage system does not have a robust self-healing system, it is difficult to guarantee its availability and scalability. The self-healing system should be able to detect failure and handle data reconstructing without human intervention.

The self-healing system of DAOS consists of two components: health monitoring system and rebuild system. DAOS is using SWIM [2], a gossip-like protocol, as the core protocol of its health monitoring system. When the health monitoring system detects failure of a storage node, it reports the failure to the highly replicated RAFT [3] based pool service, which can globally activate the second component, rebuild service, on all storage servers. The rebuild service can independently discover objects impacted by the fault by running placement algorithm against its local objects, and determine which objects have replicas or parity on the failed server. These components are scheduled for data reconstruction or replication to fallback servers in the background, even as application I/O are still inflight. Details of the self-healing system will be introduced in section-3 of this paper (Fig. 3).

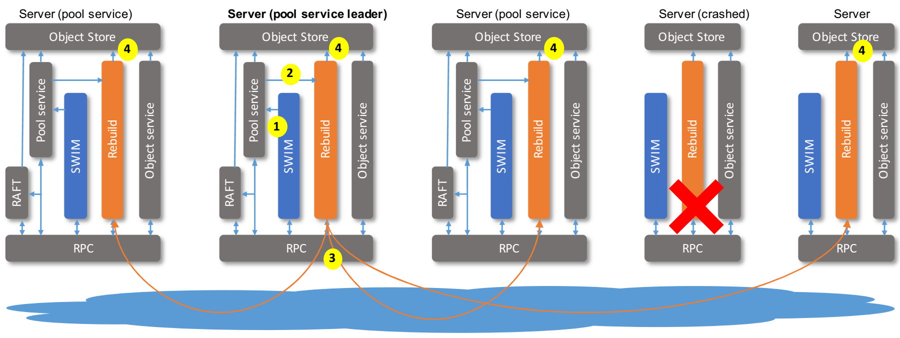


Fig. 3. Workflow of DAOS self-healing system

2 Distributed Transaction of DAOS

DAOS has both replication and erasure coding as built-in data protection strategies. Writes to an object can be distributed to multiple object shards stored on different storage nodes. Atomicity of distributed writes should be guaranteed, otherwise reads from different servers can be inconsistent and data is unrecoverable on failure. The main focus of this paper is presenting an optimized two-phase commit that can guarantee atomicity of distributed I/O while decreasing the response time of traditional protocol.

2.1 Two-Phase Commit

The two-phase commit(2PC) protocol [8] is a type of atomic commitment protocol(ACP). It is a distributed algorithm that coordinates all the members that participate in a distributed atomic transaction on whether to commit or abort the transaction. A two-phase commit transaction always needs a coordinator to drive transaction status transition among members. The coordinator can either be a dedicated process, or one of the transaction members. Within the context of this paper, transaction coordinator is also a member, it is algorithmically chosen from transaction members by running a pseudo random based function with object ID or key as random seed.

In execution of a distributed transaction, the two-phase commit protocol consists of two phases [16]:

- Prepare phase: a coordinator requests all participants to prepare for the transaction and reply vote-commit or vote-abort. If all participants voted "commit" then the transaction is "committable".
- Commit phase: based on voting of the participants, the coordinator decides whether to commit (only if all members have voted "commit") or abort the transaction, and notifies the result to all the participants.

There are a few variants of two-phase commit [9], including presumed abort(PrA), presumed commit(PrC) [5], easy commit [6], and three-phase commit [7] etc. Some of them can overcome the blocking issue of two-phase commit, others can reduce message transmission and response time of transaction, but none of them can achieve both goals.

In the case of DAOS, because distributed I/O is always tied up with data protection, so DAOS can leverage its self-healing system to support asynchronous commit and resolving the blocking issue of traditional two-phase commit. In other words, the two-phase commit introduced in this paper is a variant that is coupled with data recovery system, it is not a standalone protocol.

2.2 Asynchronous Two-Phase Commit and Batch Commit

In a basic two-phase commit protocol, the coordinator should either commit or abort the transaction before replying to client (Fig. 4.a), the response time of transaction includes two network round-trips between servers. With asynchronous commit, the coordinator can reply to the client when all members replied vote-commit for the operation (Fig. 4.b), which is called "prepared", and afterwards commit the transaction asynchronously. If any participant cannot prepare the operation, DAOS aborts the transaction synchronously.

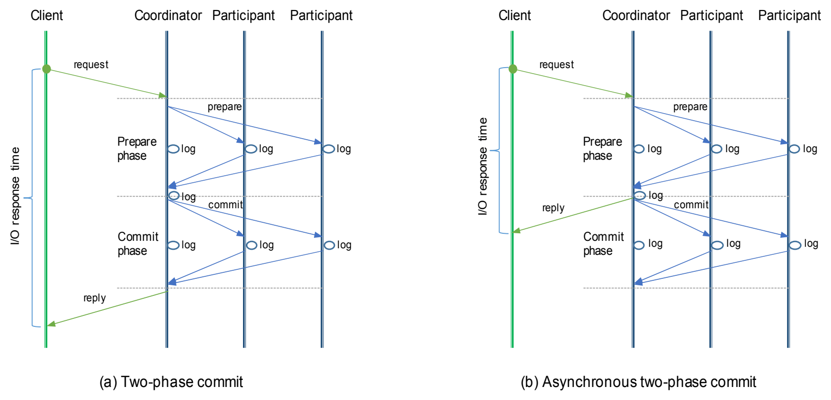


Fig. 4. Synchronous and asynchronous two-phase commit protocols

Asynchronous two-phase commit has similar response time as PrC two-phase commit protocol, but it is different with PrC in essence:

- In PrC two-phase commit protocol, coordinator should log every transaction that has started to prepare, because missing transactions are presumed to have committed. In asynchronous commit protocol of DAOS, coordinator does not log the transaction before dispatching the vote request, instead it logs the write after dispatching vote request, and other participants log the write after receiving the vote request. It means asynchronous commit protocol can save one log write and reduce the latency of transaction.
- In the asynchronous two-phase commit protocol, the logged writes on participants and coordinator are the same, they are also deemed as transaction log records. A transaction will be aborted if it is not logged by either coordinator or participant, details will be introduced in Sect. 2.5.

In the asynchronous commit protocol, transaction coordinator can reply to client before sending out the commit request. It means if clients submit many transactions against the same transaction group, the coordinator can commit them in a batch. In this approach, DAOS can significantly reduce communications between servers while also reducing persistent memory transactions by batching status changes into a single transaction. Figure 5 is an example of batched commit. In order to support asynchronously batched commit, the coordinator should cache transactions IDs that are ready to commit, or are committable, and commit them periodically or when the number of outstanding committable transactions exceeds a threshold.

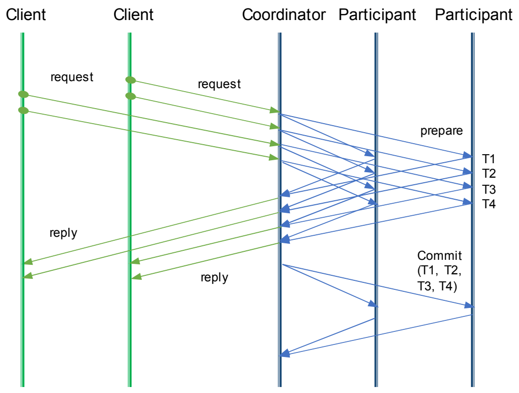


Fig. 5. Batched commit of asynchronous two-phase commit

To make this protocol practical, two issues should be addressed: 1) How a non-coordinator handles read if transaction status of data is "prepared" and 2) How to complete the transaction status transition if a member fault happened before the asynchronous commit. Solutions for these issues will be introduced in following sub-sections.

2.3 Read Protocol

Asynchronous two-phase commit of DAOS can significantly reduce latency of completing a transaction. However, it also increases complexity of the read protocol. With asynchronous commit, the writer sees write completion immediately after all members are prepared. If a reader waits long enough for the transaction to be committed asynchronously, the request can be handled normally. However, if a reader attempts to read while the asynchronous commit is in flight, the status of the transaction could be either "prepared" or "committed". In this case, it is not safe for the non-coordinator to handle the read because different servers could provide inconsistent data. So a non-coordinator should only return a special error code to the client which, instead of reporting the error to application, re-resubmits the I/O request to the coordinator that has the authoritative state of the transaction cached, either "committable" or "abort" if any members could not complete the local transaction. The coordinator can either return the correct data back to the client, or prioritize commit or abort of the transaction so other members can service reads.

2.4 Transaction Conflict

DAOS I/O can support three types of write operations: insert, update, and upsert (update or insert). Upsert of DAOS can be applied unconditionally, however, insert and update should be executed with condition check, for example, trying to insert an already existent key should fail. In order to reduce response time of RPCs, distributed I/O of DAOS does not serialize execution on primary and slaves nodes, so if two conflicting conditional operations arrived at two nodes in different order, they can end up with different execution results. In this case, both transactions should abort and restart after a random time interval until one successfully executes on all members. This paper will not include content about resolving transaction conflicts because it is a irrelevant topic, instead, the next section will introduce how a DAOS transaction proceeds if failure and conflict happen at the same time.

2.5 Non-blocking Two-Phase Commit and Transaction Resync

One of the main issues of two-phase commit protocol is that a transaction will be blocked on coordinator failure, significantly impacting availability, usability and scalability of large storage system. DAOS relies on its self-healing system, which can detect failure in bounded time and reconstruct transaction data in the background, to avoid the blocking characteristics of two-phase commit.

When a DAOS server failure happened, it can be detected by the health monitoring system (Sect. 3.1), which runs SWIM protocol, in a deterministic bound. If the coordinator was alive and received the failure event, it should return "retry" error code to the client, which can choose a fallback server to replace the failed one and re-submit the I/O transaction.

However, if the transaction coordinator failed and the storage system wants to avoid transaction blocking, then surviving members of the transaction group have to run an extra protocol to progress status of the uncommitted transaction. But if the race described in previous section and coordinator failure happen at the same time (Sect. 2.4), this process is difficult to proceed because there is no bounded time for the coordinator coming back. In the example in Fig. 6, C0/P0, which is both transaction coordinator and participant, made a different decision than other members on T1 because T1 conflicts with T0, but crashed before sending the "abort" to other members. In this case, the transaction cannot be synchronously aborted because the coordinator is gone, and nobody can even know this transaction should be aborted. In a traditional two-phase commit protocol, transaction cannot proceed before the coordinator comes back. However, bringing a server back could take unbounded time, particularly if it requires administrator, so the transaction is blocked by the failure.

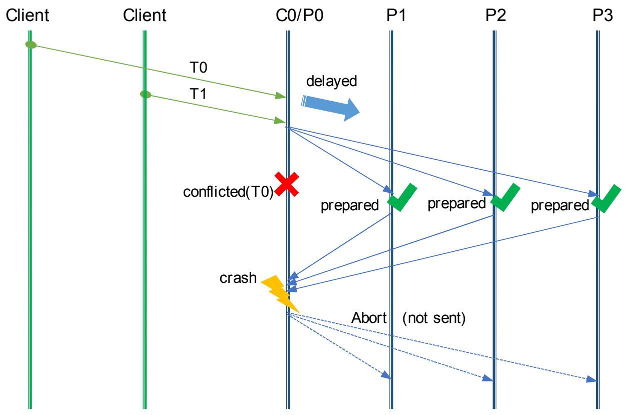


Fig. 6. Conflicting operation and transaction member failure

This is a well-known issue, DAOS resolves it by running two independent protocols: 1) resync protocol, surviving members of the transaction group should run this protocol to get agreement on status of inflight transactions, then commit or abort them; 2) rebuild protocol of self-healing system, it reconstructs data on a fallback node for all committed transactions. The rebuild protocol will be introduced in later sections, this section only focuses on resync protocol:

- If at least one of the surviving members decides to abort or has no logged vote, then the transaction group can proceed and abort the transaction, because vote of the failed one has no impact on the final decision of the transaction group.
- If the failed participant voted "abort" and it has already shared the vote with at least one of the group members, then the transaction group can also proceed and abort the transaction.
- However, if all the surviving members did vote-commit in the prepare phase, and the failed participant is the only one with "abort" vote and it crashed before sharing, the surviving members can also reach agreement and commit the transaction. It seems odd but is safe with the support of the self-healing system. Based on assumption of synchronous abort, it means neither surviving members nor client knows about the "abort" vote from the failed participant, so the self-healing system can reconstruct

data and overwrite the "abort" decision. This makes sense if the abort decision was made for an I/O error but it could also indicate a race. In the latter case, the fact that the failed participant decided to abort a transaction (C0/P0 decided to abort T1 in Fig. 6) implies others may have already decided or will decide to abort the other transaction (T0 in Fig. 6) because they already voted "prepared" for T1. So the transaction group can reach agreement and allow T1 to commit and the unseen "abort" decision will be overridden by self-healing system.

In summary, the resync protocol collects transaction votes from surviving members and makes decision without waiting for the failed member, it is not a standalone protocol because it relies on self-healing system to reconstruct committed data and even override the diverged decision. Neither transaction members nor clients will see inconsistent result with this protocol, because resync can only override abort decision if it is not known by others.

2.6 Transaction Coordinator Selection and Transaction Resync

As described in Sect. 1.3, DAOS uses pseudo-random based algorithm to generate the layout of objects. It also uses pseudo-random hash to select transaction coordinator. When a client starts an I/O against a transaction group, it can hash the object ID and map it to one of the members as the coordinator. A transactional write request has to be sent to coordinator, while read requests can be sent to any member of the transaction group, as discussed in Sect. 2.3. DAOS server uses the same pseudo-random hash to choose transaction leader, it means that for the same I/O transaction, client and servers always choose the same node as transaction coordinator.

If the transaction coordinator fails, a new coordinator must be selected by hashing object ID against and mapping to one of the surviving members. The new coordinator should immediately gather all outstanding transactions from other members, and try to commit or abort them (resync protocol), instead of caching their status in volatile memory again. This is because user data is more vulnerable after failure, those committable data should be committed so the self-healing system can reconstruct and restore the data redundancy.

The new coordinator has to be chosen from surviving members and it cannot be the fallback node in reconstructing, because only surviving members have logs for uncommitted transactions. The new coordinator can iterate log entries and request other members to move the transaction to the second phase, either commit or abort. It should be noted that some transactions might not be logged by the new coordinator, in this case they cannot be committed or aborted. It also means the original coordinator did not reply to the client, because (old) coordinator can reply only if all members confirmed "prepared" and stored the transaction in log whereas the new coordinator does not have the transaction log. So the client will eventually get a timeout from the request, and resend the request to the new coordinator and complete the transaction. If the client is also gone, then these orphan transactions will be eventually reclaimed by a background service of DAOS.

3 Self-healing System of DAOS

The self-healing system of DAOS is not just for recovering data on failure, but can also eliminate the blocking constraint of regular two-phase commit protocol. It can help the failed server to catch up transaction status when it returns, or reconstruct committed transactions on a fallback server if the original one cannot be restored. The self-healing system consists of two components: health monitoring system and rebuild protocol. This section will introduce both of them.

3.1 Health Monitoring System

DAOS uses SWIM as the health monitoring protocol. SWIM is a gossip-like protocol where node running it randomly pings a peer in the cluster and tries to share the known failures with the peer. If a node cannot reach a peer, then this node will put the peer on suspected list. After a certain timeout, if it still did not get any status update about the suspected peer, it should mark it as dead and propagate this information to other peers by random pings.

SWIM implementation of DAOS allows a server to register a notification callback, whenever a node is deemed as dead by SWIM, DAOS pool service will be notified by the callback, it can evict the dead node from the membership table, and propagate the new membership table to all nodes in the cluster. Each node receives the membership update should run "rebuild protocol" to reconstruct data for the failed node.

SWIM protocol can detect a failure in bounded time, a DAOS server running SWIM should abort message against faulty node after detecting the failure, and proceed transaction by switching to a fallback server or running resync protocol in the background, instead of blocking.

3.2 Rebuild Protocol

Rebuild protocol is the core algorithm of the DAOS self-healing system. The rebuild service of a storage node starts to run this protocol after receiving the membership update indicating a node is "down".

This protocol includes "scan" and "pull" phases. In the scan phase, a storage server scans object IDs stored in local persistent memory, independently calculates the layout of each object, and then finds out all the impacted objects by checking if the failed node is within layouts. In this phase, the rebuild service also sends IDs of these impacted object to algorithmically selected fallback servers. The fallback servers then enter the "pull" phase to reconstruct data. In this phase, fallback servers reconstruct data for impacted objects by pulling data from nodes that have redundant data of these objects, and writing the reconstructed data to the local object store.

When a storage node completes any of these two phases, it should report status to the pool service. When the pool service receives both scan and pull completions from all nodes, it can announce rebuild is globally completed by propagating the membership table again, this time the failed node is marked as "out".

As shown in Fig. 7, there is no global barrier between the scan phase and pull phase, these phases can overlap on different nodes. For example, node3 has already started to

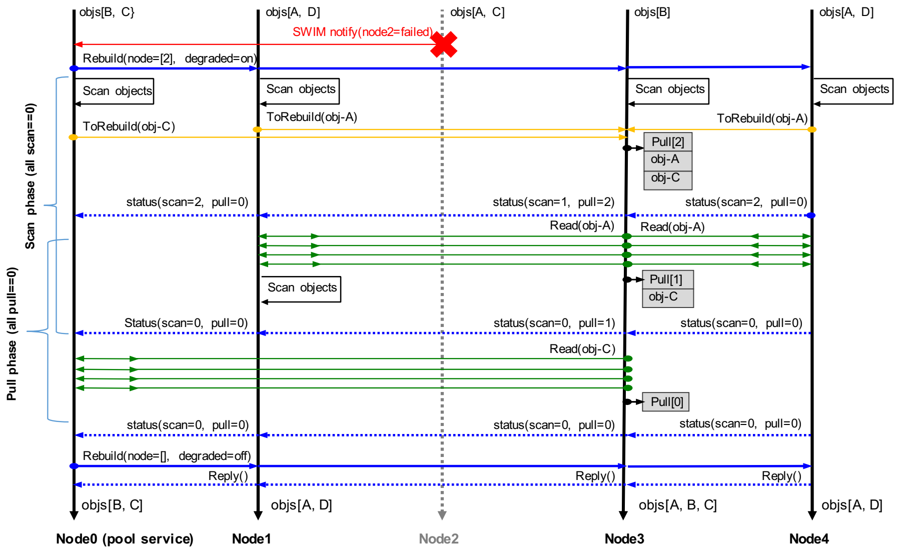


Fig. 7. Rebuild protocol of DAOS

pull data while node1 is still scanning. It means that a storage node may report false completion because it could get more object IDs after it reported "pull" completion, if a remote peer is still in scan phase and it can send object IDs to this node time to time. Therefore, the phase transition of rebuild protocol can be described like:

- A storage node should report "scan" completion once it scanned its local objects and sent out all impacted object IDs.
- A storage node should report "pull" completion each time it completed data reconstruction for all currently received objects, it means a node can report "pull" completion more than once, because after reporting, it may still receive object IDs from remote peers.
- The pool service can only trust the "pull" completions after it received all "scan" completions, because no one will provide objects for rebuild once all nodes have completed scan, then no one will report false "pull" completion anymore.

The essential of DAOS object placement algorithm is a pseudo random based hash that can distribute objects to everywhere in the storage system, so a storage node can belong to thousands or more redundancy groups. During the rebuilding process, objects impacted by the failure are distributed to nearly all the nodes, so there is no central place to perform data or metadata scans or data reconstruction. In addition, storage model of DAOS is multi-tenancy and user can create many storage pools on the same set of storage nodes, so this gives another level of rebuild declustering because objects within different pools have different layouts. In other words, the I/O workload of the rebuild service will be fully declustered and parallelized.

3.3 Cascading Failure Rebuild

Rebuild protocol of DAOS is also based on a two-phase commit protocol. Most of the work is done in the "prepare" phase that includes both "scan" and "pull". The commit phase only propagates the membership table to complete rebuild. Again, the major issue of two-phase commit is that it is a blocking protocol. DAOS is using rebuild system to eliminate the blocking of two-phase commit I/O. Since the rebuild system itself is also based on two-phase commit protocol, how can DAOS handles cascading failure without blocking the current rebuild protocol? An obvious approach is restarting the rebuild process for cascading failure where all members scan object store again to detect objects impacted by the new failure. However, in a large system with thousands of storage nodes, restarting could happen frequently because MTBF is relatively short and possibility of cascading failure is high. Tracking and resuming rebuild progress becomes a big challenge in this case in order to make progress and move to a clean status.

DAOS is using a very simple approach to avoid the blocking and restarting rebuild protocol: it simply queues the new failure, ignores all the impacts of new failure and continues the rebuild for the original failure, only handling the new one after completing the original. To explain this, two roles are defined for a storage node while running rebuild protocol:

- Contributor: a contributor should detect all local objects being impacted by the failure and is the data source for data recovery.
- Puller: a puller is the fallback node that is responsible for reconstructing data, it receives object IDs from contributors, and reconstructs data for these objects by pulling data from contributors and writing to local storage.

Although a node can be both puller and contributor (node3 of Fig. 7), they are separated in this section to simply the description. When a cascading failure happens during rebuild:

- If the newly failed node is a contributor and its data is still available on other nodes, then rebuild can proceed because other nodes can provide everything being provided by the new faulty node, the rebuild service can just switch to degraded mode and pull data from other places. On the other hand, if no other node can provide the same information as the new faulty node, it means that data is unrecoverable after cascading failure. In such cases, the rebuild protocol should also proceed because there is nothing it can do.
- If the new faulty node is a puller, then the data being reconstructed on that node is gone again but will be reconstructed by the queued rebuild task for cascading failure, so there is no necessity to start over.

Based the description above, rebuild protocol of DAOS allows the data rebuild process to proceed even there is a cascading failure, so the system would neither block nor restart the rebuild process. These characteristics ensure the protocol is scalable in a large scale storage system.

4 Asynchronous 2-Phase Commit Performance Results

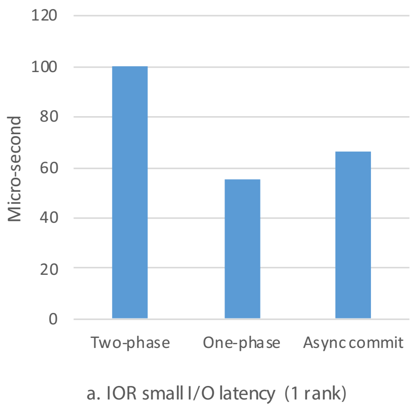
This section shows the performance differences by running IOR with and without asynchronous commit. Since one of the major goals of this protocol is reducing latency and increasing throughput of small I/O size transaction, the benchmark used 256 bytes as the transfer size to avoid the noise of bulk transfer. The results also include data points from unsafe, non-transactional or one-phase writes, which have no commit, thus the I/O is deemed as complete as long as all members are prepared.

The benchmarks have been run on Intel's DAOS prototype cluster "boro". Both client and storage nodes use Intel Xeon E5-2699 v3 processor and they are equipped with Intel Omni-Path 100 adapters. There is no persistent memory or NVMe SSD on these nodes, so data was written to tmpfs though libpmemobj of PMDK [10], which still calls flush and drain instructions even its backend is tmpfs based emulation. This does not impact the conclusion because the goal of this benchmark is showing benefits of protocol with reduction of network transmissions and cache flushes. The object in the benchmark was 3-way replicated, so the transaction group has three severs. There was a single client in the benchmark, it ran one rank for the latency test, and 16 ranks for the throughput test.

There are three bars in each part of the diagram:

- The first bar is two-phase commit that does synchronous commit, it shows the performance of the basic two-phase commit protocol.
- The second bar is one-phase distributed I/O, it skips the commit phase to represent the baseline performance of the benchmark when there is no overhead of transaction protocol.
- The third bar represents the performance result of asynchronously batch commit protocol, comparing it with the first bar can show the performance gain from running this protocol.

Figure 8.a shows I/O latency of asynchronous two-phase commit reduced 35% while comparing with regular two-phase commit, and Fig. 8.b shows small I/O throughput of asynchronous two-phase commit increased 40%.



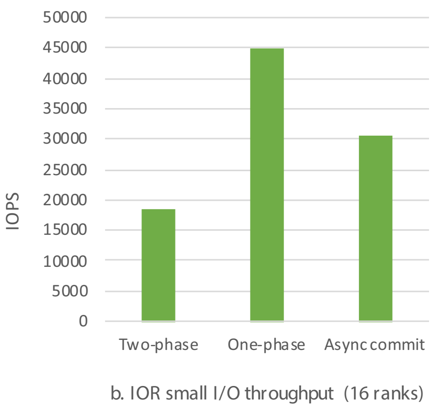


Fig. 8. IOR latency and throughput (3-way replication)

5 Conclusion

Two-phase commit protocol of DAOS is tightly coupled with its self-healing system and can avoid the unbounded blocking phenomena of a traditional implementation of two-phase commit protocol, thus increasing the availability of system. In addition, because it allows a committable transaction to move to the commit phase even in the case of multiple failures that includes both the coordinator and participant, so it can support asynchronous commit and decrease transaction response time significantly. It can also support batch commit for transactions belonging to the same transaction group, reducing the message transmissions between servers and the number of persistent memory transactions, thereby improving the overall throughput of the storage cluster.

6 Future Work

The transaction protocol introduced in this paper is only for atomicity of a replicated or erasure coding I/O against one redundancy group, it can also be extended to transactions that modifies multiple redundancy groups of arbitrary number of objects. This extension cannot simply determine transaction order by arriving order anymore but has to rely on MVCC and a global logical clock to define transaction order and control the consistency of data accessed by multiple concurrent transactions. The enhanced protocol is not described in this paper due to limited space available.

References

1. Breitenfeld, M.: DAOS for Extreme-scale Systems in Scientific Applications (2017) https://arxiv.org/pdf/1712.00423.pdf
2. Abhinandan, D., Indranil, G., Ashish, M.: SWIM: Scalable weakly-consistent infection-style process group membership protocol. In: DSN 2002 Proceedings of the 2002 International Conference on Dependable Systems and Networks. pp. 303–312 (2002)
3. Diego, O., John, O.: In Search of an Understandable Consensus Algorithm (2014) https://www.usenix.org/system/files/conference/atc14/atc14-paper-ongaro.pdf
4. Sage, A., Weil, S., Brandt, Ethan, A., Miller, L., Carlos, M.: CRUSH: controlled, scalable, decentralized placement of replicated data. In: SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing. (2006) https://doi.org/10.1109/sc.2006.19
5. Butler, L., David, L.: A new presumed commit optimization for two phase commit. In: VLDB 1993: Proceedings of the 19th International Conference on Very Large Data Bases. pp. 630–640 (1993)
6. Suyash, G., Sadoghi, M.: EasyCommit: a non-blocking two-phase commit protocol. In: International Conference on Extending Database Technologies, At Vienna, Austria (2018) https://doi.org/10.5441/002/edbt.2018.15
7. Yousef, J.A., George S.: Three-Phase Commit. Encyclopedia of Database Systems. Springer, Boston, MA (2009) https://doi.org/10.1007/978-0-387-39940-9
8. George, S., Kathryn, B., Andrew, C., Mohan, C.: Two-phase commit optimizations and trade-offs in the commercial environment. In: Proceedings of IEEE 9th International Conference on Data Engineering (1993) https://doi.org/10.1109/icde.1993.344028

9. Liu, M.L., Agrawal, D., El Abbadi, A.: The performance of two phase commit protocols in the presence of site failures. Distr. Parallel Databases **6**, 157–182 (1998)https://doi.org/10.1023/a:1008639314265
10. Andy, R.: APIs for persistent memory programming (2018). https://storageconference.us/2018/Presentations/Rudoff.pdf
11. Mercury Homepage: https://mercury-hpc.github.io/documentation/
12. Libfabric Homepage: https://ofiwg.github.io/libfabric/
13. SPDK Homepage: https://spdk.io/
14. PMDK Homepage: https://pmem.io/pmdk/
15. DAOS Homepage: https://github.com/daos-stack/daos
16. Two-phase commit Wikipedia: https://en.wikipedia.org/wiki/Two-phase_commit_protocol