2016 IEEE 22nd International Conference on Parallel and Distributed Systems

PROAR: A Weak Consistency Model For Ceph

Jiayuan Zhang*, Yongwei Wu†, Yeh-Ching Chung‡
*Graduate School at Shenzhen, Tsinghua University
Shenzhen 518057, China
Email: jiayuan-14@mails.tsinghua.edu.cn
†Department of Computer Science and Technology, Tsinghua University
Beijing 100084, China
Email: wuyw@tsinghua.edu.cn
‡Research Institute of Tsinghua University in ShenZhen
Shenzhen 518057, China
Email:yehching.chung@gmail.com

***Abstract*—The primary-copy consistency model used in Ceph cannot satisfy the low latency requirement of write operation required by users. In this paper, we propose a weak consistency model, PROAR, based on a distributed hash ring mechanism to allow clients to only commit data to the primary node and synchronize data to replication nodes asynchronously in Ceph. Based on the distributed hash ring mechanism, the low latency requirement of write operation can be met. In addition, the workload of the primary node can be reduced while that of replication nodes can be more balanced. We have evaluated the proposed scheme on a Ceph storage system with 3 storage nodes. The experimental results show that PROAR can reduce about 50% write overhead compared to that of Ceph and has a more balanced workload around all the replication nodes.**

***Index Terms*—weak consistency; Ceph; cloud storage; object storage; PROAR;**

I. Introduction

Data replications are widely used for improving availability and increasing fault tolerance of cloud systems, with the prevalence of cloud storage service over the world. The main issue of data replications is how to synchronize replications to the same state when one of them has been modified. This is known as the replication consistency problem. To handle this problem, some cloud storage systems apply a strong form of consistency to maintain the semantics of one-copy serializability [2] or linearizability [10] to ensure the correctness of each replication. However, there are also some resorting to weak consistency semantics, which permit clients change the states of some replications and make all replications into the same state after a period.

The two consistency forms have their advantages and disadvantages. In the strong consistency model, such as primary-copy [24], chain [21], etc., the replication process must ensure all replications are at the same state before any other file operations can be processed. This model, in general, will result in a high latency of write operations. Ceph [22] is a representative example of such model. For the weak consistency models, such as Dynamo [7], Cassandra [13], PNUTS [6], and Bayou [20], etc., the replication process is completed when one of the replication is updated and other file operations can be proceeded. Some data consistency mechanisms are used to ensure all replications are at the same state after a period. This model, in general, will lead to a low latency of write operations compared to that of the strong consistency model. Dynamo is a representative example of such model.

Due to the trade-off between consistency and performance in cloud storage system [4, 9], some systems start to consider combining the weak and strong consistency models by offering both services or mix some characteristics of them, the hybrid consistency model. Cloud storage systems such as Gemini [15], ALPS [16], Walter [19], Explicit consistency [3], etc., enhance the semantics of weak consistency model in order to control write-write conflict or offer a service with causal consistency [1, 14] for users. Gemini is a representative example of the hybrid consistency mode since it provides both strong consistency and weak consistency models and can choose a consistency model for a request operation automatically.

RADOS [24] is a reliable object storage service built as part of the Ceph distributed file system. It only implements the primary-copy consistency model, a kind of strong consistency. As a cloud storage system in contemporary conditions, this is not enough to satisfy the requirements of clients. Given the advantages of weak consistency model and to extend categories of Ceph consistency, in this paper, we propose a weak consistency model, PROAR, to reduce the latency of write operations of Ceph while make the workload of replication nodes more balanced. The proposed weak consistency mode has the following two contributions:

- PROAR can reduce about 50% write overhead compared to that of Ceph and has a more balanced workload around all the replication nodes.
- With PROAR, Ceph is able to provide a hybrid consistency model for applications.

The remainder of this paper is organized as follows: Section II discusses the related works. An overview of the architecture of RADOS is given in Section III. Section IV describes the PROAR consistency model in details. Section V presents the experimental results of PROAR.

1521-9097/16 $31.00 © 2016 IEEE
DOI 10.1109/ICPADS.2016.52

II. Related Work

Customized consistency model. In recent years, there are many storage systems use eventual consistency for replications, such as Openstack Swift [17]. Openstack Swift chooses quorum algorithm [8] as the basis of its consistency model, which permits clients to configure the values of the read quorum and the write quorum to get different consistency semantics. By default, Openstack Swift uses the semantic of eventual consistency. When clients write data to servers, system only commits data to $N/2$ servers, where N is the number of replications. Furthermore, when clients read data from servers, system only chooses the closest sever that contains the data as the read replica sever. The data in the selected server may be out-of-dated. In this case, clients maybe read the out-of-dated data. However, clients can configure the number of read and write replicas to setup the strong consistency or eventual consistency used in the system, according to the protocol of quorum.

Explicit consistency [3] allows applications to specify the invariants, or consistency rules, that system must maintain. Explicit Consistency is defined in terms of application properties: the system is free to reorder execution of operations at different replicas, provided that the specified invariants are maintained.

Enhanced eventual consistency model. In RedBlue consistency [15], operations are classified into two types, *blue* and *red*. Blue operations execute locally and are lazily replicated in an eventually consistent manner. Red operations, in contrast, are serialized with respect to each other and require immediate cross-site coordination. In addition, RedBlue consistency can classify different operations into red or blue automatically.

Walter [19] ensures a new isolation property called Parallel Snapshot Isolation (PSI). Across sites, PSI enforces partial causal ordering that allows the system to replicate transactions asynchronously. In order to prevent write-write conflicts, Walter relies on two technologies: preferred sites and counting sets.

ALPS [16] has implemented causal+ consistency model that is a causal consistency with convergent conflict handling. This consistency semantic is respect to the causal dependencies between operations. The conflict detection part in ALPS, which handles the conflicts of divergence of objects, ensures that clients see a causally-correct, conflict-free, and always-progressing data store.

Ceph consistency model. Ceph only offers primary-copy consistency semantic. In Ceph, the primary node executes each client operation in serialization and informs the replication nodes to execute this operation as the orders of primary node [5]. After all the replications are executed successfully, the primary node returns client the successful response code. Apparently, the consistency mechanism guarantees the strong consistency semantics.

III. Ceph Overview

Ceph has implemented frameworks including CRUSH [23], RADOS, and the communication module to store data/object. As shown in Figure 1, each object stored by the system is first mapped into a placement group (PG), a logical collection of objects that is replicated by the same set of devices. The PG of an object o is determined by a hash of the name of o, the desired level of replication r, and a bit mask m that controls the total number of placement groups in the system, that is, pgid = (r, hash(o) & m), where & is a bit-wise AND and the mask m= 2^k-1. As the cluster scales, it is necessary to adjust the total number of placement groups periodically by changing m. The changing is done gradually to throttle the resulting migration of PGs between devices.

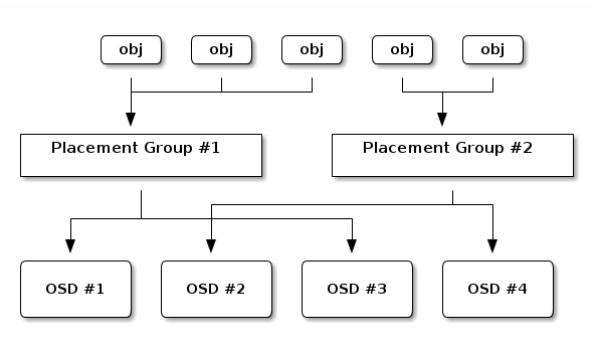

Figure 1: The procedure of objects locating to osds

PGs are allocated to object storage devices (OSDs) based on CRUSH algorithm, a robust replica distribution algorithm that calculates a stable and pseudo-random mapping. While other hash algorithms force a reshuffle of all prior mappings, in Ceph, PGs are assigned to OSDs based on the *cluster map*, a map that stores the mapping of each PG to an ordered list of r OSDs. From a high-level perspective, the behavior of CRUSH is similar to a hash function. The PGs are distributed deterministically but pseudo-randomly. Unlike a hash function, however, CRUSH is stable. When one (or many) of devices join or leave the cluster, most PGs remain where they are and only small amount of data are shifted to maintain a balanced distribution. In contrast, other hashing approaches force a reshuffle of all prior mappings. In CRUSH, it also uses weights to control the relative amount of data assigned to each device based on its capacity and loading.

The primary-copy consistency model is implemented as follows. When a client writes an object to Ceph, the client will first hash the object into a PG. Then, client sends a message to the monitor cluster to get the current cluster map, which CRUSH algorithm is used to calculate all the OSDs of the PG. Finally, client directly sends data to the primary role OSD and the primary role OSD does not responds client with ack until all replicas have been committed to disk. This procedure can refer to Figure 2 and all the operations of the same object will be directed to the same primary role OSD. Thus, it can ensure that all the operations of different clients will be executed linearly by the same primary role OSD.

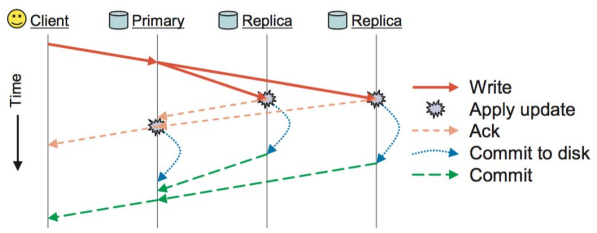

Figure 2: The procedure of RADOS handling the write execution.

IV. PROAR Consistency Model

To take the advantages of RADOS and avoid the disadvantages of primary-copy algorithm [2, 18], we propose a weak consistency model called PROAR (**Pr**imary R**o**le H**a**sh **R**ing) for Ceph. In this model, PROAR performs the following three tasks:

1) PROAR builds a primary role hash ring after PG getting all the OSDs via CRUSH algorithm;
2) PROAR requires that all the operations, including write operation and read operation, must be handled by the primary role OSD to ensure that all the operations of the same object are executed as the request orders of clients. In addition, to reduce the latency of write operation, PROAR only commits data to the primary role OSD and responses client with the result of commit;
3) PROAR stores logs of all the replication role OSDs of the PG. Logs not only ensure the primary node and replication nodes at the same state when system converges, but also can be used for the recovery of down OSD.

A. Primary role hash ring

PROAR uses the idea of hash ring to hash OSDs and objects into the same hash ring [12], which called the primary hash ring. OSDs of a PG are hashed into a 32-bit space circle as the split point in the primary hash ring. They will play as the primary role OSDs of those objects between them in the primary hash ring. As shows in Figure 3, there are three replication OSDs of all the objects in a PG. *osd.*1 is the primary role OSD of objects between *PROARHash*(*osd.* 3) and *PROARHash*(*osd.*1) in the primary role hash ring, where *PROARHash* is a function to get the hash values of OSDs in the primary role hash ring. Its details will be described later. *S*1 is the scope in between *PROARHash*(*osd.*3) and *PROARHash*(*osd.*1). The write operation, read operation, and other operations of each object in scope *S*2 will be directed to *osd.*1. Similarly, operations of objects in the scope of *S*1 will be controlled by *osd.*2. From Figure 3, we can see that all the operations of objects in the PG will be dispatched to the 3 primary role OSDs based on the hashing of objects. This will lead to a more balanced workload compare with Ceph in which all operations are handled by only one OSD.

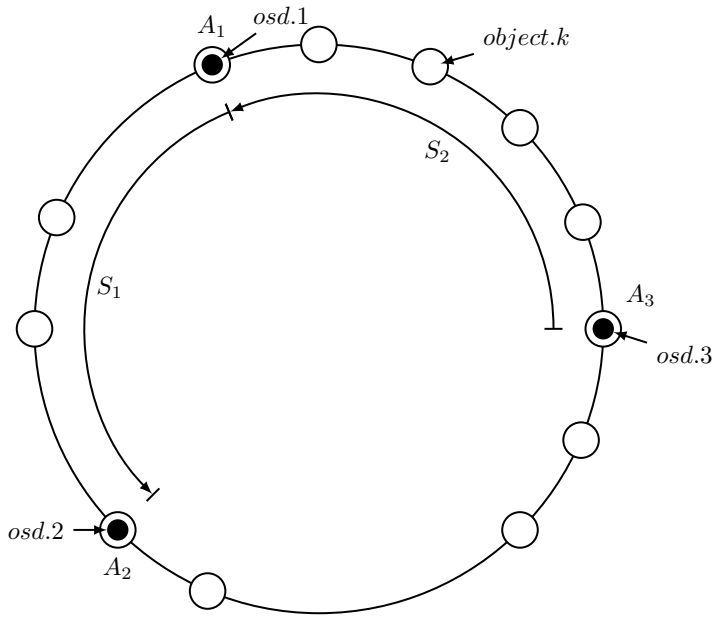

Figure 3: A PG replication circle consisting of 3 OSDs osd.1, osd.2, osd.3.

Algorithm 1 gives a pseudo code of *PROARHash*. In Algorithm 1, the input is *CRUSHArray* that is obtained by using the CRUSH algorithm and the output is *HashArray*. To get *HashArray*, this function firstly gets the even point of 32-bit ultimate value. The even point value, named *unitValue*, equals the value of *ULONG MAX*, the ultimate value of 32-bit, divided by the replica number of PG. Then the value of *unitValue* multiplying the index of each OSD is the hash value of each OSD in *CRUSHArray*.

Algorithm 1 Get the hash value of OSDs

input: $CRUSHArray$
output: $HashArray$
1: **function** PROARHash($CRUSHArray$)
2: **for** $i = 0 \rightarrow CRUSHArray.size()$ **do**
3: $unitValue =$
4: $ULONG\_MAX/CRUSHArray.size()$
5: $HashArray[i] = i * unitValue$
6: **end for**
7: **return** $HashArray$
8: **end function**

The procedure for clients to locate the corresponding primary role OSD for giving objects in the hash ring is as follows:

1) The CRUSH algorithm is used to get the replication OSDs, which is an OSD array called *CRUSHArray*.
2) Client uses *PROARHash* function to get the hash value array, named *HashArray*, of *CRUSHArray*.
3) Client compares the object with all the values in *HashArray* to get its scope and get the corresponding primary role OSD.

For example, if *client*1 wants to write data to *object*1, it uses CRUSH algorithm to get *CRUSHArray*, which format is like [1,0,2]. Next, *client*1 uses *PROARHash* function to get *HashArray*, and compare the *object*1.id with all the values of *Hasharry*. If *HashArray*[*i*] <*object*1.id < *HashArray*[*i*+1], the primary role OSD of *object*1 is *osd.i*. *PROARHash* takes advantages of CRUSH algorithm and hashes OSD to the invariant value according to their locations in the *CRUSHArray*. Hence, if an OSD of the PG is down, *PROARHash* can hash the backup OSD to the same value into the primary role hash ring. This reduces the work to reshuffle objects to another OSD.

In summary, in Ceph, PGs are hashed to different OSDs by CRUSH algorithm. The first OSD in the OSD array is selected as the primary node and other OSDs are replication nodes. The primary node will handle all the object operation requests from clients. Hence, the primary node suffers more workloads in compare with replication nodes. In the primary role hash ring scheme, *PROARHash* can make the workloads of the primary role OSD and replication role OSDs more balanced since all OSDs will play the primary role for some objects. As shown in Figure 1, objects in PG #1 are located in *osd.*1 and *osd.*3. Under the primary-copy consistency rules, all the objects are handled by *osd.*1. In PROAR, the primary role OSD of objects in the same PG is split into *osd.*1 and *osd.*3. *osd.*3 will take the primary role for some objects in the PG and can relax the work pressure of *osd.*1.

On the other hand, OSD workload balance can also reduce the latency of write operations. If there are many write operations to a single primary OSD, most of them will be blocked on the waiting queue. With multiple primary role OSDs presented in PROAR, this problem can be relaxed and the waiting time can be reduced.

B. Single primary node write

It is the most important method to reduce the latency of write operation for Ceph. But this method will bring another problem, that is, data on primary node and replication data copies will divergent. For users, it cannot be tolerant that if the data read is out-of-dated. Hence, in *rule* 2 as mentioned before, the read OSD of clients is restricted. Like write operation, read operation also uses PROAR and *PROARhash* to locate the primary OSD and only gets data from the primary node. In this way, although it can not reduce the latency of read, it ensures the client will read the updated data.

To reduce the latency of write operation, in PROAR, we use the *single primary role OSD write* policy to implement the weak consistency model. In this policy, once an updated data is written to disk by the primary role OSD, all operations can be proceeded even the replication role OSDs have not yet completed the writing of their replications. This policy is the most efficient method to reduce the latency of write operation in Ceph. However, this method also suffers that a read operation may get an out-of-dated data. To eliminate this issue, in PROAR, a read operation only get data from the corresponding primary role OSD, not replication role OSDs. In this way,although it can not reduce the latency of read, it ensures that the data read by a client is up-to-date.

The single primary role OSD write procedure is given as follows:

1) Client uses CRUSH algorithm to get *CRUSHArray*;
2) According to *CRUSHArray*, get the corresponding primary role OSD;
3) Client writes data to the primary role OSD. If operation is successful, primary role OSD responses the client with successful answer, else return false.

To compare with Ceph, PROAR reduces the time of sending data to replication node, waiting for the replication node to execute the operation, and data writing back to disk.

C. Convergence

PROAR uses PG log to ensure the synchronization between the primary role OSD and the replication role OSDs. Each PG maintains the primary role OSD log and the replication role OSDs log. In the primary role OSD, when each operation is executed successfully, system callback is invoked to write the operation entry to the PG log of this primary role OSD. As shown in Figure 4, the PG log entry format contains 4 columns, which are labeled as *op*, *object*, *osd* and *eversion*. Column *op* stores the operation type of clients. In this column, M represents modify, D means delete, and R denotes read. Column *object* stores the object name. Column *osd* contains the primary role OSD location in the CRUSH output array. Column *version*, which increments itself after the primary role OSD executes a client operation such as write, read or delete. Parameter *last_commit* points to the entry that has been synchronized to replications successfully. Parameter *last_update* points to the entry that has been updated. After synchronizations are completed successfully, *last_commit* and *last_update* will point to the same entry.

In addition, when receiving data from primary role OSD at the synchronization period, the replication OSD adds log entry to the PG log after it stores all the data from the primary role OSD and set *last_commit* and *last_update* to the corresponding primary role OSD log as the newest entry.

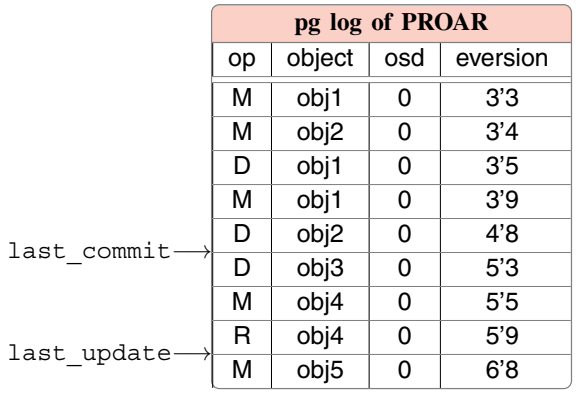

Figure 4: Entry format of pg log in PROAR.

Convergence procedure: The convergence procedure is shown in Algorithm 2. In Algorithm 2, parameter *log_map* is a map that stores PG log of each OSD. Parameter *peers* indicates all the OSDs in this PG. Parameter *whoami* is the identification (ID) of an OSD. When the synchronization process starts, each replication role OSD of PG first gets its ID via parameter *whoami*. Then each replication role OSD uses parameter *whoami* as a key to get the primary role log of this PG. When the replication role OSD get the objects from the primary role OSD, it writes the log entry to log table of the primary role OSD. Finally, the replication role OSD informs the primary role OSD with results of execution. Eventually, all the OSDs of this PG converge to the same state with the same operation orders.

When an OSD is down, PG will recalculate the replication role OSDs through CRUSH algorithm, and then use *PROARHash* to recalculate the hash values of OSDs in the primary role hash ring. *PROARHash* will hash the backup OSD to the same value of the down OSD.

Algorithm 2 Replication Convergence

input: $log\_map, peers, whoami$
output: $NUll$

```
1:  function RepConverg(log_map, peers, whoami)
2:      primary_role_log = log_map.get(whoami)
3:      start = primary_role_log.last_commit.index()
4:      end = primary_role_log.last_upate.index()
5:      for j = start → end do
6:          for i = 0 → peers.size() do
7:              if peers[i] == whoami then
8:                  continue
9:              end if
10:             response = 0
11:             osd_con_msg =
12:             new osd_con_msg(peers[i],
13:             primary_role_log.getEntry(j), objectdata
14:             , result)
15:             send(peers[i], osd_con_msg)
16:             if result then
17:                 response + +
18:             end if
19:         end for
20:         if response == peers.size() − 1 then
21:             last_commit + +
22:         end if
23:     end for
24: end function
```

V. Performance evaluation

This section presents an evaluation of PROAR. We will compare PROAR with the primary-copy consistency model used in Ceph in terms of the write latency and the workload balancing of OSDs.

A. Experimental setup

Unless stated otherwise, experiments are conducted on a cluster with 4 Intel NUCs: 1 monitor and 3 OSDs. The profile of Intel NUC is shown in the table below:

CPU	RAM	HDD
Intel Core i7	4G DDR3	1TB 5400rpm

Figure 5 shows the architecture of the cluster. In Figure 5, we can see that all the servers are connected under the same LAN, where *mon.a* is the monitor of the cluster and others are OSDs. The speed of the network is 14.6 Mbits/sec, which is measured by iperf [11]. The throughout of HDD is 54.5 MB/s for each OSD machine. To evaluate the impact of PROAR, we have the following three settings for Ceph:

Setting 1: Ceph with the primary-copy consistency model, 333PGs, and 0 replication;

Setting 2: Ceph with the primary-copy consistency model, 333PGs, and 3 replications; and

Setting 3: Ceph with PROAR consistency model, 333 PGs, and 3 replications.

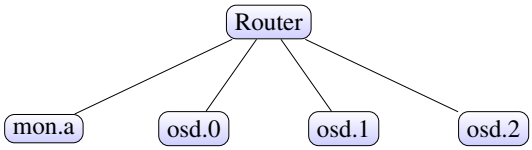

Figure 5: the topology of experiment cluster

To verify the impact of PROAR and the single node write for the write latency, we create 1 replicated pool with 3 replications, 1 replicated pool with no replication and PROAR pool with 3 replications.

B. Write Latency

Figure 6 shows the performance of the write latency of the three settings with 1800 write requests and data size at 4M, 8M, 16M, 32M and 64M bytes. From Figure 6, we can see that the PROAR can reduce about 10% and 50% latency of write operation compared to that of Ceph with 0 and 3 replications, respectively. The proposed method has lower write latency compared with Ceph with 0 replication is due to the distributed scheme of the primary role OSDs used in PROAR. This scheme can reduce the waiting time of write operations when dispatching to the primary role OSD for execution. With the data size increased, the improvement of write latency of PROAR is more obvious.

In compare with 3x replication, PROAR has reduced the write operation latency sharply , it is because that PROAR only commits data to the primary role OSD and return user with the execution result. Whereas, 3x replication mode has to commit data to all the OSDs.

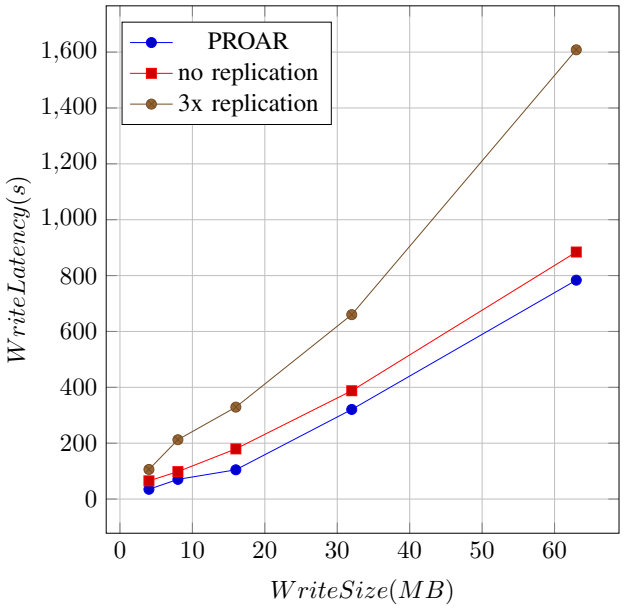

Figure 6: Write latency for varying write sizes and replication.

C. OSD Workloads Balance

Figure 7 shows the workload performed by each OSD of the three settings with 1800 write requests. From Figure 7, we can see that the average workload and standard deviation of Settings 2 and 3 are (600, 29.02) and (600, 520.56), respectively. The PROAR has a more balanced workload compared to that of Ceph. The reason, again, is due to the distributed scheme of the primary role OSDs used in PROAR.

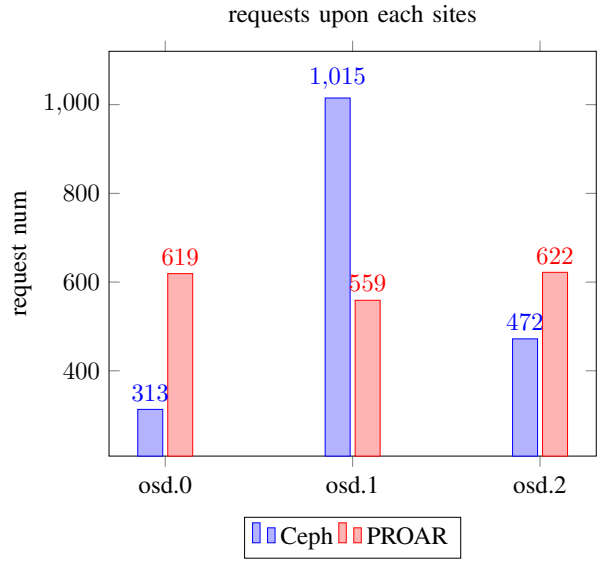

Figure 7: The requests of each OSDs in a pool

VI. Conclusion

This paper proposes a novel consistency model, PROAR, for Ceph. The key feature behind PAROAR is the primary role hash ring of a PG. Based on the primary role hash ring, PROAR spreads objects in a PG into different replication nodes, permits single node write, and makes the system converge to the same state. We have implemented PROAR in Ceph and expand the pool categories of Ceph to allow users to different consistency models for their applications.

Acknowledgment

We appreciate the work of Ceph community, who offers an awesome system selflessly and provide a base for our work. The work of this paper is partially supported by Shenzhen City Branch Committee under contract No. 2016-092.

References

[1] Ahamad, Mustaque et al. “Causal memory: Definitions, implementation, and programming”. In: *Distributed Computing* 9.1 (1995), pp. 37–49.

[2] Alsberg, Peter A and Day, John D. “A principle for resilient sharing of distributed resources”. In: *Proceedings of the 2nd international conference on Software engineering*. IEEE Computer Society Press. 1976, pp. 562–570.

[3] Balegas, Valter et al. “Putting consistency back into eventual consistency”. In: *Proceedings of the Tenth European Conference on Computer Systems*. ACM. 2015, p. 6.

[4] Brewer, Eric A. “Towards robust distributed systems”. In: *PODC*. Vol. 7. 2000.

[5] *ceph architecture*. http://docs.ceph.com/docs/hammer/architecture/. Accessed: 2016-07-01.

[6] Cooper, Brian F et al. “PNUTS: Yahoo!’s hosted data serving platform”. In: *Proceedings of the VLDB Endowment* 1.2 (2008), pp. 1277–1288.

[7] DeCandia, Giuseppe et al. “Dynamo: amazon’s highly available key-value store”. In: *ACM SIGOPS Operating Systems Review* 41.6 (2007), pp. 205–220.

[8] Gifford, David K. “Weighted voting for replicated data”. In: *Proceedings of the seventh ACM symposium on Operating systems principles*. ACM. 1979, pp. 150–162.

[9] Gilbert, Seth and Lynch, Nancy. “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services”. In: *ACM SIGACT News* 33.2 (2002), pp. 51–59.

[10] Herlihy, Maurice P and Wing, Jeannette M. “Linearizability: A correctness condition for concurrent objects”. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 12.3 (1990), pp. 463–492.

[11] *iperf homepage*. https://iperf.fr/. Accessed: 2016-07-01.

[12] Karger, David et al. “Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web”. In: *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. ACM. 1997, pp. 654–663.

[13] Lakshman, Avinash and Malik, Prashant. “Cassandra: a decentralized structured storage system”. In: *ACM SIGOPS Operating Systems Review* 44.2 (2010), pp. 35–40.

[14] Lamport, Leslie. “Time, clocks, and the ordering of events in a distributed system”. In: *Communications of the ACM* 21.7 (1978), pp. 558–565.

[15] Li, Cheng et al. “Making geo-replicated systems fast as possible, consistent when necessary”. In: *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 2012, pp. 265–278.

[16] Lloyd, Wyatt et al. “Don’t settle for eventual: scalable causal consistency for wide-area storage with COPS”. In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM. 2011, pp. 401–416.

[17] *openstack swift*. http://docs.openstack.org/developer/swift/. Accessed: 2016-07-01.

[18] Saito, Yasushi and Shapiro, Marc. “Optimistic replication”. In: *ACM Computing Surveys (CSUR)* 37.1 (2005), pp. 42–81.

[19] Sovran, Yair et al. “Transactional storage for geo-replicated systems”. In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM. 2011, pp. 385–400.

[20] Terry, Douglas B et al. “Managing update conflicts in Bayou, a weakly connected replicated storage system”. In: *ACM SIGOPS Operating Systems Review*. Vol. 29. 5. ACM. 1995, pp. 172–182.

[21] Van Renesse, Robbert and Schneider, Fred B. “Chain Replication for Supporting High Throughput and Availability.” In: *OSDI*. Vol. 4. 2004, pp. 91–104.

[22] Weil, Sage A et al. “Ceph: A scalable, high-performance distributed file system”. In: *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association. 2006, pp. 307–320.

[23] Weil, Sage A et al. “CRUSH: Controlled, scalable, decentralized placement of replicated data”. In: *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM. 2006, p. 122.

[24] Weil, Sage A et al. “Rados: a scalable, reliable storage service for petabyte-scale storage clusters”. In: *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing’07*. ACM. 2007, pp. 35–44.