Alternative Reliability Models in Ceph

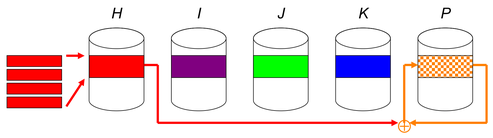
David Bigelow, Scott Brandt, Carlos Maltzahn
{dbigelow, scott, carlosm}@cs.ucsc.edu

Abstract

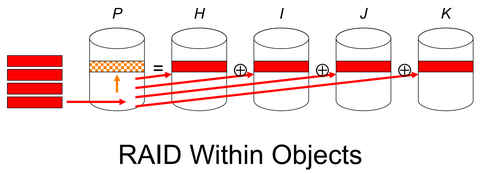
RAID systems have traditionally offered increased performance and data security in small storage systems. An opportunity now exists to extend traditional RAID principles into the area of large-scale object-based storage devices in order to offer greater data security and space efficiency. In a system where component failures can be expected on a daily basis, the importance of redundancy mechanisms is obvious, and RAID principles offer an appropriate model. Ceph is an excellent platform with which to test these RAID principles, and learn how they function in a new environment. However, there are several details that need examination before a full implementation can be done within the Ceph system.

RAID Across and Within

- The Orange Checkered block represents the parity object, written to the parity disk *P*.
- The Red block is a data object that the user wishes to write. It can be broken up into smaller objects; four in this case.
- The Purple, Green, and Blue blocks represent data unrelated to the current write.
- The arrows represent the flow of data on any write.



RAID Across Objects



RAID Within Objects

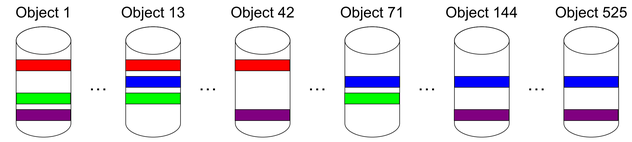
RAID Across Objects is the equivalent of doing only small writes in a normal system. Each time a user wishes to write an object to the storage system, they direct their write to a single device which sends updated parity information to the parity device for that group. Reading has normal behavior. This scheme can be extended to any error-correcting code that maintains one “normal” copy of the data, and uses any number of parity/recovery disks. RAID Across has the following attributes, assuming a simple xor-based scheme:

- Writes are fast. Data only needs to be written on two devices (or *n* devices for a more complicated scheme).
- Reads are fast. Data is only read from a single known device, which does not need to consult any other to return the data.
- The required bandwidth is exactly the size of the object, plus some amount of overhead (acks, etc.)
- Two full sets of parity calculations are necessary. The primary disks must first xor the old object with the new one, and then send it to the parity disk, which xors this new object with its current parity block.
- If the parity group is operating in degraded mode (in other words, device *H* is unavailable), both reading and writing are slow and depend on the load of the other devices. Data flow can be directed at any of the remaining devices (*I, J, K, P*), but all must be consulted to recover data.

RAID Within Objects is the equivalent of doing only large writes in a standard RAID system. Writes and reads are both directed at the parity disk, which does the work of distributing (or gathering) the data by breaking the original object into several pieces. This scheme can also be extended to any error-correcting code which maintains one “normal” copy of the data and uses any number of parity/recovery disks. Obviously, all the data in a single group is related. RAID Within has the following attributes, assuming a simple xor-based scheme:

- Writes are slow. Data needs to be written on every device on the group, and the data cannot be confirmed written until the slowest one returns.
- Reads are slow. Since reading is directed at the parity disk (which maintains none of the actual data), all data must be reassembled before it can be returned to the user. This might be negated if the client were smart enough to read their data directly from all the devices in the group simultaneously.
- The required bandwidth is exactly the size of the object, plus some amount of overhead (more than RAID Across, since more devices are communicating).
- Only one set of parity calculations must be performed at the initial disk.
- If the group is operating in a degraded mode (any of the disks being unavailable), then writes are marginally faster and reads are marginally slower. However, if the parity disk itself has failed, the client must be smart enough to write appropriate sub-objects to the other devices in the group.

General RAID-Based Reliability Scheme



The colored boxes each represent one object in a parity group. Each object is the same size, but parity groups may be of arbitrary size.

This is the most general model of a reliability scheme utilizing RAID-based principles. No assumptions are made on how parity groups are arranged, and nor do they necessarily have to be parity-based. The sole limitation is that each object must be the same size within its group. In the example given above, the data might be arranged as: The red blocks are duplicates of the same object, utilizing three way mirrors. The blue blocks are arranged in a traditional 3+1 parity scheme commonly found in RAID. The green blocks are in a 2+1 parity scheme. The purple blocks are arranged in a 2+2 Reed-Solomon type encoding. This could all be done within the same storage system with an appropriate hierarchical model.

For a general model, there are several properties:

- Parity groups do not need to be shared over multiple object sets. In other words, if one parity group is defined by the set {1,13,42,71,144,525}, there may not exist any other group over that exact set of devices. Another set might be {1,42,57,113,181,325}, having only two devices in common with the first.
- One object set has no bearing on any other object set. Therefore, a mix of encoding schemes can be done over different object sets, so long as one is able to determine what scheme is associated with each group.
- Parity groups may contain objects of arbitrary size, so long as all objects within a group are the same. One group might be composed of kilobyte-sized objects, another of four-megabyte sized objects.

	RAID Across	RAID Within
Write Speed	Fast	Slow
Read Speed	Fast	Slow or Fast (depending on client)
Required Calculations	2x	1x
Amount of time spent in degraded mode	Very little	Slightly more than RAID Across
Performance in degraded mode	Poor	Good
Required changes to Ceph	Many	Few

Points to Consider

- Because object sizes are so variable, the traditional advantages of RAID parallelization may not apply. Any client can gain this advantage on its own by simply breaking its writes into additional objects. If only one object is ever read at a time, break it into smaller objects and read them all simultaneously.
- All of these schemes are based around not trusting the client and having the storage devices do all the work. However, there may be times when it is appororiate to trust the client to do its own calculations.

Current Status

- Currently working on a simulation of several different RAID-like schemes and error-correcting codes.
- Schemes being considered include RAID-4 and RAID-5 like encoding schemes, special encoding schemes such as EVENODD, error-correcting codes such as Reed-Solomon schemes, all being evaluated over the RAID Across and RAID Within models.
- Goal of determining bandwidth costs, computational costs, and gathering performance metrics.
- The next step is to complete a Ceph implementation of at least one of the schemes, depending on what the simulations reveal.
- RAID Across seems to have many advantages in every category except degraded performance mode, but it is much harder to integrate into the existing Ceph framework.
- RAID Within offers lower performance in general (butt higher performance when a device has failed), and is much easier to integrate into the existing Ceph framework.

Future Work

- Implement all components of a working RAID model into Ceph. This includes basic read/write functionality, failure mode operation, failure recovery, and automatic group rebuilding. These components can be added piecewise, but must be present for a full implementation.
- Of secondary importance is the inclusion of a hierarchical mode which allows multiple reliability modes (as described in the general model above). This would allow mirroring for data which requires high-performance, and RAID-like modes for less urgent data.
- It might also be worthwhile to explore the possibility of using the client to calculate parity for their own data. This would only work in certain modes of operation, but would relieve the burden of computation from the storage system devices.