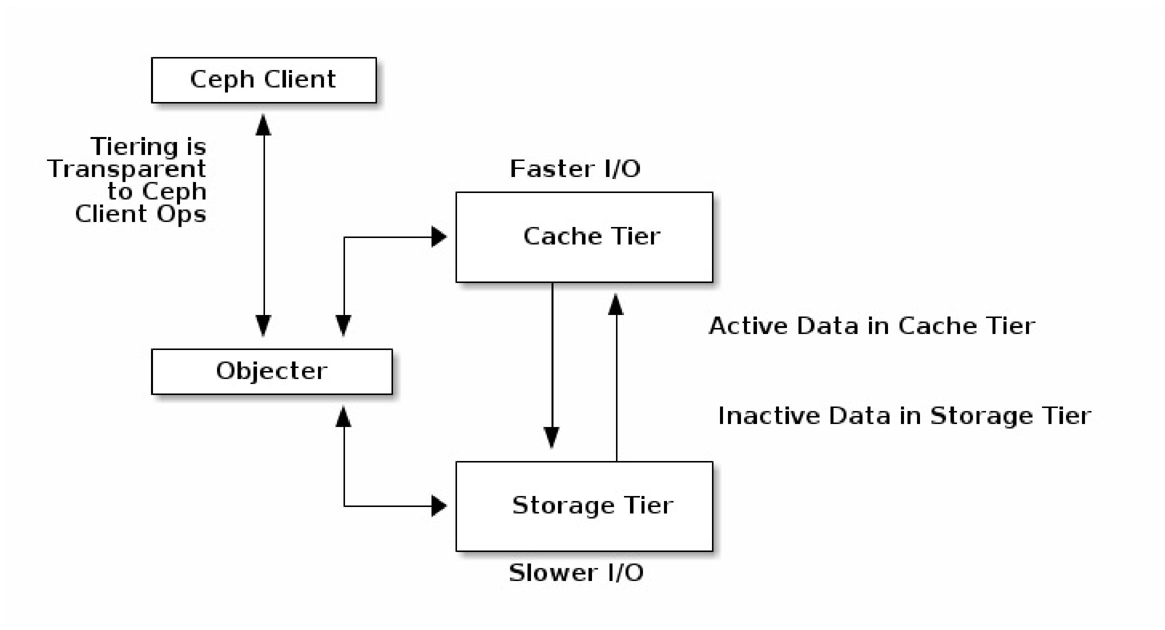
CACHE TIERING

A cache tier provides Ceph Clients with better I/O performance for a subset of the data stored in a backing storage tier. Cache tiering involves creating a pool of relatively fast/expensive storage devices (e.g., solid state drives) configured to act as a cache tier, and a backing pool of either erasure-coded or relatively slower/cheaper devices configured to act as an economical storage tier. The Ceph objecter handles where to place the objects and the tiering agent determines when to flush objects from the cache to the backing storage tier. So the cache tier and the backing storage tier are completely transparent to Ceph clients.


Ceph Client
Tiering is Transparent to Ceph Client Ops
Faster I/O
Cache Tier
Active Data in Cache Tier
Objecter
Inactive Data in Storage Tier
Storage Tier
Slower I/O

The cache tiering agent handles the migration of data between the cache tier and the backing storage tier automatically. However, admins have the ability to configure how this migration takes place. There are two main scenarios:

- **Writeback Mode:** When admins configure tiers with `writeback` mode, Ceph clients write data to the cache tier and receive an ACK from the cache tier. In time, the data written to the cache tier migrates to the storage tier and gets flushed from the cache tier. Conceptually, the cache tier is overlaid “in front” of the backing storage tier. When a Ceph client needs data that resides in the storage tier, the cache tiering agent migrates the data to the cache tier on read, then it is sent to the Ceph client. Thereafter, the Ceph client can perform I/O using the cache tier, until the data becomes inactive. This is ideal for mutable data (e.g., photo/video editing, transactional data, etc.).
- **Read-proxy Mode:** This mode will use any objects that already exist in the cache tier, but if an object is not present in the cache the request will be proxied to the base tier. This is useful for transitioning from `writeback` mode to a disabled cache as it allows the workload to function properly while the cache is drained, without adding any new objects to the cache.

A WORD OF CAUTION

Cache tiering will *degrade* performance for most workloads. Users should use extreme caution before using this feature.

- *Workload dependent*: Whether a cache will improve performance is highly dependent on the workload. Because there is a cost associated with moving objects into or out of the cache, it can only be effective when there is a *large skew* in the access pattern in the data set, such that most of the requests touch a small number of objects. The cache pool should be large enough to capture the working set for your workload to avoid thrashing.
- *Difficult to benchmark*: Most benchmarks that users run to measure performance will show terrible performance with cache tiering, in part because very few of them skew requests toward a small set of objects, it can take a long time for the cache to “warm up,” and because the warm-up cost can be high.
- *Usually slower*: For workloads that are not cache tiering-friendly, performance is often slower than a normal RADOS pool without cache tiering enabled.
- *librados object enumeration*: The librados-level object enumeration API is not meant to be coherent in the presence of the case. If your application is using librados directly and relies on object enumeration, cache tiering will probably not work as expected. (This is not a problem for RGW, RBD, or CephFS.)
- *Complexity*: Enabling cache tiering means that a lot of additional machinery and complexity within the RADOS cluster is being used. This increases the probability that you will encounter a bug in the system that other users have not yet encountered and will put your deployment at a higher level of risk.

KNOWN GOOD WORKLOADS

- *RGW time-skewed*: If the RGW workload is such that almost all read operations are directed at recently written objects, a simple cache tiering configuration that destages recently written objects from the cache to the base tier after a configurable period can work well.

KNOWN BAD WORKLOADS

The following configurations are *known to work poorly* with cache tiering.

- *RBD with replicated cache and erasure-coded base*: This is a common request, but usually does not perform well. Even reasonably skewed workloads still send some small writes to cold objects, and because small writes are not yet supported by the erasure-coded pool, entire (usually 4 MB) objects must be migrated into the cache in order to satisfy a small (often 4 KB) write. Only a handful of users have successfully deployed this configuration, and it only works for them because their data is extremely cold (backups) and they are not in any way sensitive to performance.
- *RBD with replicated cache and base*: RBD with a replicated base tier does better than when the base is erasure coded, but it is still highly dependent on the amount of skew in the workload, and very difficult to validate. The user will need to have a good understanding of their workload and will need to tune the cache tiering parameters carefully.

SETTING UP POOLS

To set up cache tiering, you must have two pools. One will act as the backing storage and the other will act as the cache.

SETTING UP A BACKING STORAGE POOL

Setting up a backing storage pool typically involves one of two scenarios:

- **Standard Storage**: In this scenario, the pool stores multiple copies of an object in the Ceph Storage Cluster.
- **Erasure Coding:** In this scenario, the pool uses erasure coding to store data much more efficiently with a small performance tradeoff.

In the standard storage scenario, you can setup a CRUSH rule to establish the failure domain (e.g., osd, host, chassis, rack, row, etc.). Ceph OSD Daemons perform optimally when all storage drives in the rule are of the same size, speed (both RPMs and throughput) and type. See CRUSH Maps for details on creating a rule. Once you have created a rule, create a backing storage pool.

In the erasure coding scenario, the pool creation arguments will generate the appropriate rule automatically. See Create a Pool for details.

In subsequent examples, we will refer to the backing storage pool as `cold-storage`.

SETTING UP A CACHE POOL

Setting up a cache pool follows the same procedure as the standard storage scenario, but with this difference: the drives for the cache tier are typically high performance drives that reside in their own servers and have their own CRUSH rule. When setting up such a rule, it should take account of the hosts that have the high performance drives while omitting the hosts that don't. See Placing Different Pools on Different OSDs for details.

In subsequent examples, we will refer to the cache pool as `hot-storage` and the backing pool as `cold-storage`.

For cache tier configuration and default values, see Pools - Set Pool Values.

CREATING A CACHE TIER

Setting up a cache tier involves associating a backing storage pool with a cache pool

```
ceph osd tier add {storagepool} {cachepool}
```

For example

```
ceph osd tier add cold-storage hot-storage
```

To set the cache mode, execute the following:

```
ceph osd tier cache-mode {cachepool} {cache-mode}
```

For example:

```
ceph osd tier cache-mode hot-storage writeback
```

The cache tiers overlay the backing storage tier, so they require one additional step: you must direct all client traffic from the storage pool to the cache pool. To direct client traffic directly to the cache pool, execute the following:

```
ceph osd tier set-overlay {storagepool} {cachepool}
```

For example:

```
ceph osd tier set-overlay cold-storage hot-storage
```

CONFIGURING A CACHE TIER

Cache tiers have several configuration options. You may set cache tier configuration options with the following usage:

```
ceph osd pool set {cachepool} {key} {value}
```

See Pools - Set Pool Values for details.

TARGET SIZE AND TYPE

Ceph’s production cache tiers use a Bloom Filter for the `hit_set_type`:

```
ceph osd pool set {cachepool} hit_set_type bloom
```

For example:

```
ceph osd pool set hot-storage hit_set_type bloom
```

The `hit_set_count` and `hit_set_period` define how many such HitSets to store, and how much time each HitSet should cover.

```
ceph osd pool set {cachepool} hit_set_count 12
ceph osd pool set {cachepool} hit_set_period 14400
ceph osd pool set {cachepool} target_max_bytes 1000000000000
```

Note: A larger `hit_set_count` results in more RAM consumed by the `ceph-osd` process.

Binning accesses over time allows Ceph to determine whether a Ceph client accessed an object at least once, or more than once over a time period (“age” vs “temperature”).

The `min_read_recency_for_promote` defines how many HitSets to check for the existence of an object when handling a read operation. The checking result is used to decide whether to promote the object asynchronously. Its value should be between 0 and `hit_set_count`. If it’s set to 0, the object is always promoted. If it’s set to 1, the current HitSet is checked. And if this object is in the current HitSet, it’s promoted. Otherwise not. For the other values, the exact number of archive HitSets are checked. The object is promoted if the object is found in any of the most recent `min_read_recency_for_promote` HitSets.

A similar parameter can be set for the write operation, which is min_write_recency_for_promote.

```
ceph osd pool set {cachepool} min_read_recency_for_promote 2
ceph osd pool set {cachepool} min_write_recency_for_promote 2
```

Note: The longer the period and the higher the min_read_recency_for_promote and min_write_recency_for_promote``values, the more RAM the ``ceph-osd daemon consumes. In particular, when the agent is active to flush or evict cache objects, all hit_set_count HitSets are loaded into RAM.

CACHE SIZING

The cache tiering agent performs two main functions:

- **Flushing:** The agent identifies modified (or dirty) objects and forwards them to the storage pool for long-term storage.
- **Evicting:** The agent identifies objects that haven't been modified (or clean) and evicts the least recently used among them from the cache.

ABSOLUTE SIZING

The cache tiering agent can flush or evict objects based upon the total number of bytes or the total number of objects. To specify a maximum number of bytes, execute the following:

```
ceph osd pool set {cachepool} target_max_bytes {#bytes}
```

For example, to flush or evict at 1 TB, execute the following:

```
ceph osd pool set hot-storage target_max_bytes 1099511627776
```

To specify the maximum number of objects, execute the following:

```
ceph osd pool set {cachepool} target_max_objects {#objects}
```

For example, to flush or evict at 1M objects, execute the following:

```
ceph osd pool set hot-storage target_max_objects 1000000
```

Note: Ceph is not able to determine the size of a cache pool automatically, so the configuration on the absolute size is required here, otherwise the flush/evict will not work. If you specify both limits, the cache tiering agent will begin flushing or evicting when either threshold is triggered.

Note: All client requests will be blocked only when target_max_bytes or target_max_objects reached

RELATIVE SIZING

The cache tiering agent can flush or evict objects relative to the size of the cache pool(specified by target_max_bytes / target_max_objects in Absolute sizing). When the cache pool consists of a certain percentage of modified (or dirty) objects, the cache tiering agent will flush them to the storage pool. To set the cache_target_dirty_ratio, execute the following:

```
ceph osd pool set {cachepool} cache_target_dirty_ratio {0.0..1.0}
```

For example, setting the value to 0.4 will begin flushing modified (dirty) objects when they reach 40% of the cache pool's capacity:

```
ceph osd pool set hot-storage cache_target_dirty_ratio 0.4
```

When the dirty objects reaches a certain percentage of its capacity, flush dirty objects with a higher speed. To set the `cache_target_dirty_high_ratio`:

```
ceph osd pool set {cachepool} cache_target_dirty_high_ratio {0.0..1.0}
```

For example, setting the value to `0.6` will begin aggressively flush dirty objects when they reach 60% of the cache pool's capacity. obviously, we'd better set the value between dirty_ratio and full_ratio:

```
ceph osd pool set hot-storage cache_target_dirty_high_ratio 0.6
```

When the cache pool reaches a certain percentage of its capacity, the cache tiering agent will evict objects to maintain free capacity. To set the `cache_target_full_ratio`, execute the following:

```
ceph osd pool set {cachepool} cache_target_full_ratio {0.0..1.0}
```

For example, setting the value to `0.8` will begin flushing unmodified (clean) objects when they reach 80% of the cache pool's capacity:

```
ceph osd pool set hot-storage cache_target_full_ratio 0.8
```

CACHE AGE

You can specify the minimum age of an object before the cache tiering agent flushes a recently modified (or dirty) object to the backing storage pool:

```
ceph osd pool set {cachepool} cache_min_flush_age {#seconds}
```

For example, to flush modified (or dirty) objects after 10 minutes, execute the following:

```
ceph osd pool set hot-storage cache_min_flush_age 600
```

You can specify the minimum age of an object before it will be evicted from the cache tier:

```
ceph osd pool {cache-tier} cache_min_evict_age {#seconds}
```

For example, to evict objects after 30 minutes, execute the following:

```
ceph osd pool set hot-storage cache_min_evict_age 1800
```

REMOVING A CACHE TIER

Removing a cache tier differs depending on whether it is a writeback cache or a read-only cache.

REMOVING A READ-ONLY CACHE

Since a read-only cache does not have modified data, you can disable and remove it without losing any recent changes to objects in the cache.

1. Change the cache-mode to none to disable it.

   ```
   ceph osd tier cache-mode {cachepool} none
   ```

   For example:

```
ceph osd tier cache-mode hot-storage none
```

2. Remove the cache pool from the backing pool.

   ```
   ceph osd tier remove {storagepool} {cachepool}
   ```

   For example:

   ```
   ceph osd tier remove cold-storage hot-storage
   ```

REMOVING A WRITEBACK CACHE

Since a writeback cache may have modified data, you must take steps to ensure that you do not lose any recent changes to objects in the cache before you disable and remove it.

1. Change the cache mode to `forward` so that new and modified objects will flush to the backing storage pool.

   ```
   ceph osd tier cache-mode {cachepool} forward
   ```

   For example:

   ```
   ceph osd tier cache-mode hot-storage forward
   ```

2. Ensure that the cache pool has been flushed. This may take a few minutes:

   ```
   rados -p {cachepool} ls
   ```

   If the cache pool still has objects, you can flush them manually. For example:

   ```
   rados -p {cachepool} cache-flush-evict-all
   ```

3. Remove the overlay so that clients will not direct traffic to the cache.

   ```
   ceph osd tier remove-overlay {storagetier}
   ```

   For example:

   ```
   ceph osd tier remove-overlay cold-storage
   ```

4. Finally, remove the cache tier pool from the backing storage pool.

   ```
   ceph osd tier remove {storagepool} {cachepool}
   ```

   For example:

   ```
   ceph osd tier remove cold-storage hot-storage
   ```