Using Cooperative Mechanisms and Replication to Improve Block-level Caching Reliability

Salma Rodriguez Jorge Cabrera

Florida International University
{*srodr063, jcabr020*}*@fiu.edu*

December 6, 2012

Device Mapper Cache

What is it?

A generic block-level caching mechanism for storage networks

What does it do?

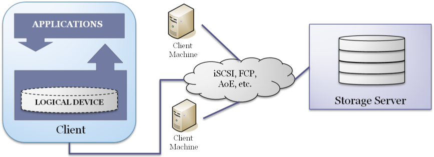
Cache popular data locally to reduce network and storage server latency

How does it work?

Built upon the Linux kernel device mapper, which maintains the mapping between a source device, and a cache device.

Device Mapper Cache

DM Cache takes advantage of spatial and temporal locality by caching data locally and using LRU for cache replacement.



Device Mapper Cache

- Distributed shared storage systems (SAN, iSCSI, AoE, &c.) support better scalibility by using block-level caching on the client side
- Fast mass storage devices, like Solid State Drives (SSDs) are excellent candidates for cache devices
- While DRAM can increase throughput by supporting more IOPS, SSD caches provide greater capacity

Replication

One-to-many distribution of data from a *source* to several *targets* to hold replicas of original data

Pessimistic Replication

- synchronous
- consistent, but not optimal: system may incur performance bottleneck from unpredictable network and storage latency
- low availability: replicator will block until data is fully propagated to all targets

Optimistic Replication

- asynchronous
- high availability, low latency, not consistent
- less reliable than pessimistic replication

Cooperative Caching

Basic Idea

- Collaboration across a network in order to saturate the space that is available for caching
- Use IP-based interface for cache devices to communicate

Sub-optimal Implementation

Use iSCSI. Works at the block layer. Only need a logical cache partition to hold replicas.

Better Implementation

Have kernel modules interact at the block layer through TCP protocol. iSCSI network interface adds unnecessary file system indirection.

Problem Statement

Facts to Consider

- Local caching: better I/O performance with write-back policy, but no redundancy (not reliable)
- Write-through policy: better reliability guarantees but lacks performance gains obtained from buffering writes on a local cache
- Client failure or cache device failure may result in critical data loss

Question

How can we improve reliability on the cloud while taking advantage of the performance gains and energy efficiency of write-back policies?

Proposed Approach

- Want to increase reliability while maintaining a reduced load on the storage server
- Implement cross-client cooperative caching and replication policies
- Replicate uncommitted data of a client cache to a logical partition of a peer cache

Framework

- Modify an existing block-level caching solution
- Use block request mapping and redirection mechanism
- Add components for cooperative caching and replication
- Add components to handle failure and recovery

Failure & Recovery

- DM Cache keeps the source device in the bi bdev of each block I/O, with block addresses being mapped to the cache device
- Need to partition the physical cache and set up each logical cache
- Each logical cache maps block addresses from the source device independently
- Already have source device. Need to flush data when replicator is not available
- Server maintains redundancy through RAID setup but may still keep data when not available and flush later

Policy for Failure Handling

Assumption: not all replicas are consistent.

Algorithm 1 Flushing the data on failure.

```
procedure DO FAILOVER
    D ← server disk
    C ← replicator cache
    T ← time specified by user
    while true do
        if C and D available then sleep for T seconds
        else                                ▷ One of C or D are not available
            V ← list of bios (latest copy)
            if D is not available then
                keep V until server disk is available
                flush V back to source device
            else                            ▷ C is not available
                flush V back to source device
```

Evaluation

Cases to Consider

- Server is not available
- Node with original data (replicator) is not available
- In both cases, the data is flushed back to the disk

Evaluating Client

To test client reliability, disconnect client node and wait for data to be flushed back to the disk

Evaluating Server

To test server reliability, disconnect server node and reconnect. Wait for blocks to be copied back and flushed by replicator.

Progress

Current Modifications

- Added a mechanism for sending block I/Os to more than two caches
- Added a device mapper parameter for choosing devices that are shared through iSCSI, which DM Cache will use to replicate data of local cache.

Pending Modifications

- Still need to implement the mechanism for flushing dirty blocks when the replicator and storage server disk are no longer available.
- Need to implement recovery policy

Challenges

How to prevent failure handling mechanism from corrupting data when flushing back to disk.

Demonstration

And now for the demo.