Using Cooperative Caching and Replication to Improve Reliability

Salma Rodriguez Jorge Cabrera
Florida International University
{srodr063, jcabr020}@fiu.edu

Abstract

1 Introduction

Client-side block-level caching has been proven to relieve storage area network systems from load balancing and scalability issues[1][6]. By using fast storage devices such as flash-based SSD devices as a local caching medium, clients can support better I/O performance compared to just using the storage system network. Despite the I/O performance benefits of a storage caching layer, reliability issues can arise if dirty data is not persisted to the storage backend frequently. On the one hand, write-back policies offer great performance benefits as they may take advantage of requests with temporal locality and reduce the load on the server. However, if the flushing of dirty data is postponed for too long, a client failure may result in cirtical data loss. Other policies such as write-through offer better reliability guarantees but discard any performance gains obtained from buffering writes. Our solution consists of a block-level caching system that implements cross-client cooperative caching and replication policies, to compromise reliability and I/O performance gains.

The remainder of this paper is organized as follows. In section 2 we provide background information on replication and cooperative caching. Section 3.2 describes our proposed approach for establishing reliability using cooperative caching and replication. Finally, section 4 contains a brief schedule of the tasks that will take place to implement and test our cooperative caching and replication code.

2 Background

Replication techniques for conflict resolution have been explored for distributed systems where multiple users require concurrent access to data on a single serve node, as well as increasing the availability of data by using optimistic replication[3]. Although previous research has focused on improving reliability, availability, and consistency on distributed or grid systems, reliability for client-based caching of storage server data has had little improvement.

Pessimistic replication is a synchronous replication technique that guarantees a higher degree of consistency and reliability by blocking access to data until it can be verified that the data has been replicated across the network. However, pessimistic replication increases the variance in access time, since network behavior is highly unpredictable. In the case of DM-Cache, using pessimistic replication would result in higher latency when there are more clients cooperating as replica targets. The scalibility problem with pessimistic replication is shown in [5], as well as [4]. The work of Saito and Shapiro on optimistic replication[3] does not focus on reliability, but it shows how to achieve eventual consistency on distributed systems where access to data is possible from server-class machines that are distributed across different locations on a network. The nearest location is used to obtain a consistent copy of the data, while updates are sent to a master site in charge of replicating the data. Our focus for this paper is not to achieve eventual consistency, but we will use optimistic replication in order to maintain high availability and high throughput. The goal is to increase reliability by using cooperative caching and replication, without affecting performance and availability. Therefore, optimistic replication is the best choice for our approach.

In [2], replication schemes and their apact on system reliability are explored. Two replica optimizers are compared and combined into a third to increase data availibility for files of different sizes. The paper by Lei et al. propose online optimization algorithms to replicate data based on a file weight. The weight depends on the popularity of the file, its availability, the number of copies, as well as its size. The weight is largely affected by the

popularity of the file. The hotter the file, the higher the possibility of it being replicated. Availability improves by allowing files that are not popular to be replaced by replicated copies of more popular data. Although reliability improvements are not emphasized in the paper, it can be assumed that by making the data highly available, there is an increase in reliability. The paper motivated us to consider the number of replicas per client in our approach for improving reliability using cooperative caching with DM Cache.

3 Approach

We will use optimistic replication in our approach, which is comprised of implementing a block-level caching system that replicates a client's dirty data across the caches of other clients. Our approach is to use a cooperative caching protocol to make clients in the network aware of other peers' cache capacity and utilize any free storage available to propagate replicas of dirty data. In order to compensate for the speculative approach of replicating updates across clients, we consider the tradeoff between free space and the degree of replication in our implementation of an algorithm that is guaranteed to maximize the number of clients with replicate copies. This will help ensure system-wide reliability. For single-client reliability, a simple greedy algorithm is enough to propagate the data to clients with enough free space to store a replica upon updates.

3.1 Proposed Design

For this project, the assumption is that clients are part of a storage area network (SAN), and that they use an IP-based storage network solution such as iSCSI to communicate with the backend storage server. Our prototype will consist of a network of peer clients using iSCSI as their communication protocol. We chose iSCSI as the communication protocol because it already implements an effective mechanism that allows machines to move blocks of data across each other. In the current system, where clients work in a non-cooperative environment, the storage server functioned as the iSCSI target, and the client machine functioned as the isCSI initiator. In other words, the client machine would only access the storage resources allocated for it in the server. In order to transfer blocks of data from client to client, the new system will require that all clients see a portion of their peer clients as targets. Thus in the new system clients will take the role of both initiators, such as when they propagate data to their clients, and may also take the role of targets such as when they become the recipients of replicated data.

3.2 Implementation

The implementation of our prototype consists of using DM-Cache, an example of an existing block-level caching solution, and modifying it to incorporate our cooperative caching and replication mechanisms. The current implementation of DM-Cache exists as a Linux kernel module. It works by intercepting block requests directed towards the main storage device, and redirecting these towards a storage cache device. Information about redirected requests such as source device to cache device block mappings, block status flags, and other related meta-data is kept in a special data structure. Our prototype will use DM-Cache, its block request mapping and redirection mechanism as the foundation of the block-level cache management mechanism.

In order to implement the cooperative caching mechanism we plan to use our customized version of the iSCSI initiator and target code. Current implementations of iSCSI initiator manage login and discovery of target functions in user space, and they use a kernel-space application to handle the data movement functions. For our initial prototype, the plan is to use only the data movement portion of the iSCSI initiator and target, and implement the login and discovery of peer client targets on the kernel side as a helper module to DM-Cache. The reason for this is that iSCSI is designed to allow the initiator machine to see and use the target machine as a local device. We do not want this functionality as it may allow the client to modify or corrupt a peer client's cache.

When loading our version of DM-Cache on a client machine, it will have the chance to specify a circle of peer clients to which it can propagate replicas. A circle is basically a group of clients which have agreed to let another peer client use its free cache space. A client needs only the IP, the port, and the iSCSI identifier of all the clients that are part of the circle in order to connect to them. When a propagation request is activated, a client will send out a request to each client in its circle and it will wait for each client to respond with a notification, letting it know whether it has any space available in its cache. Once all the clients in the circle have responded, a subset of these will be chosen to receive a replica of the dirty data.

Our cooperative caching mechanism will allow a client to specify its circle of clients only before loading DM-Cache on it. The reason for doing this in a static fashion rather than dynamic is because we first want to focus on creating a communication channel that is stable and that has acceptable performance in comparison to the baseline case. If time allows it, a best-case scenario implementation will result in a system allowing a client to join a network of cooperative clients without having to reload DM-Cache with a new circle of clients.

In our design, a *captured page* is a dirty page belonging to another client. Captured pages are pinned to avoid accidental swapping or deletion. The *degree of replication* is equivalent to the number of captured pages in a client. A *replication candidate* is a client with enough free space in its local cache to satisfy a replication request. Clients replicate data proactively in the background. Our asynchronous approach allows the data to be available immediately after propagation is initiated. Our algorithm considers the tradeoff between free space and the *degree of replication* in each client in order to perform fair replication across clients, and ensure system-wide reliability. A kernel thread will run in the background for each client. The client with the original data with notify other clients via sockets when pages are flushed back to the disk. Once a flush notification is recieved by the client holding the read-only replica, the captured pages will simply be deleted from the cache. The benefit for choosing a replication candidate with enough free space is as follows:

$$benefit = \frac{f_i}{r_i}$$

where f_i is the free space in node i, and r_i is the degree of replication in node i.

In our approach we keep a time stamp of every replica. Every time an update request reaches a client with the replica of an original copy, the client will locate the replica in its local cache, and update the captured pages as well as the time stamp. Whenever a failure occurs, after bootup, the iSCSI kernel module will load the neighboring clients, and once block-level access is established for each client, the recovery process will begin. The iSCSI kernel module probes each of the neighboring clients for lost data that was not successfully written back to disk during the previous session, and once the data is found, it will be copied back to the local cache, and the read-only flags will be dropped from the reclaimed dirty pages.

4 Schedule

4. Schedule Phase 1 (September 30 - October 13) Start the implementation initial prototype: Jorge will work on setting up a simple communication mechanism that allows a single client to send block requests to client machine. The goal is to start working with a modified version of the iSCSI framework. Salma will work on the implementing the logic to replicate a portion of client's block on another machine in dm-cache. Phase 1 Presentation Slides (October 10): Both Salma and Jorge put together a presentation of their project Phase 2 (October 13 - December 3) Expand the functionalities on the initial prototype: Jorge will add the functionality to the communication protocol to allow a single client to send requests to multiple client machines. Salma will work on adding the algorithm to trigger the propagation of a clientŠs dirty data to other client machines Phase 2 Report (November 11): Both Salma and Jorge work on the putting together a report on their project's status Work on finalizing customized iSCSI communication protocol: Jorge will work finalizing the code of the iSCSI which allows a clients to communicate to be both an initiator and a target for other clients Finalizing replication policy algorithm: Salma will work on adding logic to manage replicas belonging to different clients Testing final version of dm-cache: Both Salma and Jorge Put together the final version of the cooperative dm-cache Perform final tests on prototype Prepare demo for final presentation Final presentation (December 3 - December 16) Present the demo of cooperative dm-cache that replicates a client's data on peer client machines Write final report: - December 12

References

[1] HENSBERGEN, E. V., AND ZHAO, M. Dynamic Policy Disk Caching for Storage Networking. Tech. Rep. RC24123 (W0611-189), IBM Research, November 2006.

[2] LEI, M., VRBSKY, S. V., AND ZIJIE, Q. Online Grid Replication Optimizers to Improve System Reliability. In *Proceedings of the International Parallel and Distributed Processing Symposium* (2007).

[3] SAITO, Y., AND SHAPIRO, M. Optimistic Replication. *ACM Computing Surveys, Vol. 37, No.1* (2005).

[4] YU, H., AND VAHDAT, A. Minimal Replication Cost for Availability. In *21st Symposium on Principles of Distributed Computing* (2001).

[5] YU, H., AND VAHDAT, A. The Costs and Limits of Availability for Replicated Services. In *18th Symposium on Operating Systems Principles* (2001).

[6] ZHAO, M. Visa research lab: dm-cache. Accessed September 29, 2012.