University of Wisconsin Madison

Cluster File Cache

Author:
Nathan Mitchell

Supervisor:
Dr. Miron Livny

July 8, 2011

1 Overview

The Cluster File Cache will be an expansion from the original cluster file transfer project, building on the former's file transfer mechanism and adding distributed file replication within the cluster.

1.1 Motivation

While the original goals of the Cluster File Transfer project are effectively being migrated to the Cluster File Cache, the reason behind this seeming name change is as follows:

Thus far the progress under the original title has resulted in a working point to point transfer protocol for cluster file management. While this fits the old project title well, it does not accomplish the overall focus of the project, namely robust file placement within a cluster environment. Thus the new project will swing focus directly to that issue while making use of the previously developed transfer protocol to handle data transfer.

1.2 System Components

The general design of the new system is inspired heavily from existing distributed content delivery systems, namely the bit-torrent protocol. While this system will be making use of the CFT protocol instead of bit-torrent, there are other features of modern bit-torrent systems that can be put to use here:

Content Description Records Referred to as torrent files, these documents contain information about a collection of files to be distributed, including identifying hash values for consistency checks.

Robust Metadata Storage While originally bit-torrent required centralized repository of the previously mentioned torrent files, modern bit-torrent systems can make use of DHT's or distributed hash tables to act as distributed, efficient, and robust data storage[2].

These two features form the core of the new Cluster File Cache system proposed here. The CFC will be composed of Cache Nodes, CN's, running on every node of the cluster. They will be connected via an overlay network, within which will exist a DHT containing Cache Descriptors and File references.

A Cache Descriptor will be a record that contains information about a set of Files to be contained within the Cache. This will include basic information about the files themselves, for example file names and hash values, plus replication requirements.

2 System Operation

2.1 Components

2.1.1 Files

A File is the smallest item stored by the CFC (smallest defined here by atomicity, not storage space) and contains data to be stored by the system. A File is represented by a Hash in the system and a Node will never store anymore than one copy of a File locally. The Hash of a File is made from the file's contents.

2.1.2 Hash

A Hash is an identifying signature that can represent a Node, a Descriptor, or a File within the system. Any implementations should use care to chose sufficiently strong hash functions to make collisions rare.

2.1.3 Cache Descriptor

A Cache Descriptor, or Descriptor, is a record that references a set of Files for replication. It contains the following parts:

Title A textual identifier

Creation Date The time-stamp the Descriptor was first created. This value is immutable and will never change as long as the Descriptor exists within the Cache.

Update Date The time-stamp of when the Descriptor was last altered. This value allows Nodes to keep track of changes to Descriptors within the Cache.

File List The list of files referenced by this Descriptor. For purposes of replication, all the files in this list must exist at a Node for the Descriptor to be considered replicated to that Node, otherwise there is at most a partial replication.

Replication Restrictions A list of requirements for replication to occur. Currently these include the following (subject to expansion and change as this document evolves):

1. Node - Only Nodes that mach this hostname will replicate this Descriptor
2. Min Count - Nodes will continue to replicate this Descriptor until the Cache contains at least this number of full replicates. Implementations may decide if further replication will occur or not.

The Hash of a Descriptor is constructed from its Title and Creation Date.

2.1.4 Cache Node

A Cache Node, or Node, is a singe instance of a caching service running within the cluster. It maintains a File repository, Descriptor repository, and participates in the DHT. The Hash of a Node is dependent on the implementation of the DHT, but typically consists of the Node's IP address and additional information.

2.1.5 Distributed Hash Table

The Distributed Hash Table[1][3][4], DHT, is a globally shared repository shared, and maintained, between all the Nodes. It contains the following information about the Cache status:

Descriptor Registry This record lists all the Hash's of currently registered Descriptors and the their current replication count. Nodes are responsible for periodically retrieving the registry and confirming their copy is up to date. This registry is updated whenever a Descriptor is added or permanently removed from the Cache, or when a Node alters the replication status of the Descriptor.

Descriptors Descriptor records are retrieved from the DHT via their Hash values, which are recorded in the Registry. The location of the any Descriptor is not fixed as they are replicated within the DHT when they are created or updated.

File Indexes File Indexes are retrieved from the DHT by the Hash of the file they represent and contain the list of all Node Hashes which contain that File in their File repository. These indexes are updated whenever a Node either gains or loses a File.

2.1.6 Cache Access Point

The user facing interface to a Cache Node. Via this interface client, a user may query the status of the Cache as a whole, or at a per Node basis. Additionally, the user may add/delete Descriptors and Files or update existing Descriptors. Finally, the Access Point provides a mechanism to retrieve locally stored Files from a Node.

2.2 Operation

2.2.1 Initiation

Initially the Cache is empty. The first Node that is brought up in the cluster is considered the 'joining' or 'bootstrap' Node. Any subsequent node deployed on the cluster can access the Cache via this Node (technically any Node could perform this function), but as the Cache may be deployed dynamically, any external master starting service may find it convenient to keep track of the first Node as a initialization parameter for the rest.

Regardless, as soon as the first Node is initialized, the Descriptor Registry is added to the DHT, albeit empty. The next step for any Node is to scan its File repository and record into the DHT any Files it contains into File Indexes. Depending if the implementation supports it, a similar step may be preformed for Descriptors, entering any preexisting local copies into the Descriptor Registry and thus the DHT.

2.2.2 Descriptor Addition

If a Descriptor is entered into a Node, it first checks to see if the Descriptor already exists within the Registry, and if so is an update, or not.

1. If the Descriptor is new, the Node updates the Registry and then proceeds to store the Descriptor in the DHT.

2. If the Descriptor is not new, but not an update, no further action is taken.

3. If the Descriptor is not new, but is an update, the updated Descriptor is stored into the DHT at the its Hash value, which is the same as the old one.

2.2.3 Descriptor Removal

If a Descriptor is removed from the Node, it checks to see if the Descriptor already exists within the Registry or not.

1. If the Descriptor exists, it is removed from the Registry.
2. If the Descriptor does not exist, no action is taken

2.2.4 Update Interval

Periodically, each Node performs an update action. During this phase, the Node retrieves the Descriptor Registry and determines if there are any additions or removals since the last update. It then records all changes to its internal registry.

For each Descriptor in the Node's local registry, it then retrieves the Descriptor from the DHT, and confirms that it no changes have occurred. For any Descriptor with updates or if its local repository indicates less than full replication, the Node then preforms a Descriptor Evaluation.

2.2.5 Garbage Collection

Periodically, each Node performs a garbage collection action. During this phase, the Node moves through each Descriptor in its local registry and determines if it is storing any Files not referenced by any known Descriptor. If any errant Files are found, they are deleted from the Node's File repository.

2.2.6 Descriptor Evaluation

This action occurs whenever the Node examines a Descriptor in order to determine if it need s to replicate the Files referenced. For each of the replication requirements in the Descriptor, the Node determines if it can satisfy them. If this is true for all the requirements and the Node is currently not replicating the Descriptor, it will attempt to do so. In this case it will perform a Descriptor Replication. Otherwise, no further action for this Descriptor will taken at this time.

2.2.7 Descriptor Replication

This action occurs when the Node determines it should replicate a Descriptor but is not currently. For each File in the Descriptor, it performs the following:

1. Check if the File is already stored locally, if so skip to the next File.

2. Query the DHT for the File Index and retrieve the locations this File is stored at. If the File is not in the DHT, skip to the next File.

3. For each location, attempt to initiate a transfer from that location to local storage. If a transfer is successful, update that File's Index in the DHT. Continue to next File at first success or if all attempts fail

At the end of this process, there are three outcomes:

1. Full Replication - All Files were successfully replicated.

2. Partial Replication - Some Files were replicated, but others were not.

3. No Replication - No Files were successfully replicated locally.

This status is recorded in the Node's local registry for future update intervals.

References

[1] Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J., and Stoica, I. Towards a common api for structured peer-to-peer overlays. In *Peer-to-Peer Systems II*, M. Kaashoek and I. Stoica, Eds., vol. 2735 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003, pp. 33–44.

[2] Maymounkov, P., and Mazières, D. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems* (London, UK, 2002), IPTPS '01, Springer-Verlag, pp. 53–65.

[3] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. pp. 149–160.

[4] Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D., and Kubiatowicz, J. D. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications 22* (2004), 41–53.