Mumfie DFS

Background and Idea, version 0.1

Rasmus Holm, carho647@student.liu.se

Last modified: 2012-04-25

Table of content

Mumfie Distributed File System - Background and Idea

Background

Traditional file systems such as, ext2,3,4 FAT or NTFS, has some clear limitations for end user in regard to data security, scalability, authentication , fixed location and so on. Many of this problems are generally solved with expensive hard and software solutions such as raid or extensive backup systems. This file system operates on the kernel level and the OS take care of file operations this way. The basic structure of file systems has served users well in the enviorment they were develop, were all data and resources were in one specific location(one computer or server in general). We have used this structure in One-User to One-File-System.

Today we see user level solutions as Windows Active Directory or NFS which purpose is to abstract away the file system and storage for the end user to a server or a storage network. This offers an efficient way to for user access while keeping administrative and hardware costs down, however they still suffer from the limitations of traditional file systems(ext3 and FAT) and extensive backup systems are still required and actively maintained. This is a Many-Users to One-File-System solution.

We are moving in to an era of cloud computing where more and more functions such as storage and raw computing capacity is moved from the end user and in sens abstracted away to a cloud(We see EC2, Gmail and so on). A Many-Users to Many-File-Systems solution(in seance). This is a pleasant general structure of storing data and brings great benefits but the actual software is often not open nor free and are developed for extremely big structures in a non generic way. This makes it hard for smaller companies to deploy and they have to relay on the preexisting structures (as Amazone EC2 or Dropbox).

The notion of distributed file systems is nothing new and they have been around for long but are really not available to most. There are a few that hold and have held great promise such as Hadoop and Wula. Wula was however bought from Google by LaCia and there idée of a peer-to-peer distributed file system held great promise for end-users but was not really practical for a business setting. Under LaCia's regime Wula has been commercialised into a Dropbox-like product not utilizing its potential. Hadoop is however a truly great stack of software developed for distributed computing by Google and Yahoo. It is now a open source software released under the Apache Software Foundation. Hadoop dose contain a distributing file system (Hadoop File system, HDFS) for dealing with the files when making computations. HDFS has a great strure but is made for large scale computing and has a minimum file size of 64mb and is optimal with files size between 1gb to 1tb. This however makes it fairly useless for general computing data storage.

Project Idea

To create a user-level distributed files system for general computing data storage with data replication, easy and fast access and a client API enabling an easy way to develop client application outside of the file system(ex. using fuse to enable it to mount). This type of file system can bring many benefits to smaller storage networks etc. rendering backups redundant, utilizing existing hardware to a grater extent and bringing the benefits of the cloud to the average Joe.

HDFS provides a good basic structure and model for data distribution, replication and client access. Thus making it use full as a model on which to build this file system on. The working name Mumfie is from a magical elephant from a novel as a reference to Hadoop which is the name of a toy elephant.

The Client API will provide a simple way of accessing the file system similar to a regular file accessing API in Java.

Example retrieve file:

1. The Client sends the request for a file to the Name Node.
2. The Name Node finds meta-data for requested file and returns it to the Client.
3. The Client reads the meta-data and which contain information at what Data Nods the requested file can be found
4. The Client then sends a request for the current file to one of the Data Nods, if it fails to respond the Client tries another Data Node from the meta-data list.
5. When the Data Node retrieves the request it simply returns the file.