Mumfie DFS

Requirements Specification, version 0.2

Rasmus Holm, carho647@student.liu.se
Samuel Trennelius, samtr396@student.liu.se

Last modified: 2012-02-27

Summary

MDFS is a file system that offers an alternative to traditional file systems such as FAT32, ext4 and NFS. This system ensures data security by having all files replicated on to multiple Data Nodes. If one server crashes the files wont be lost. When implemented no server need extreme high bandwidth to accommodate all data transfers. Instead the Client accesses the files on the Data Nodes directly, making it faster and more robust than NFS and similar network file systems.

MDFS contains three main parts. A Client API that enables third part software to utilize MDFS. A Name Node that contains all meta-data for the file system and users. Data Nodes that store the raw data on the local generic file system. This stack of software work together to enable general file system operations over a distributed system of nodes.

Table of content

Introduction

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 environment 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(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 and expansive for smaller companies to deploy systems like this 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 structure 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 purpose computing data storage.

Concept

The idée is to create a user-level distributed files system for general computing data storage with data replication with easy, fast and direct access to the files. MDFS will only have a client API access. In simpler terms this means to create a couple of softwares that lets a client store and access data on an undefined number of data nodes in a network or over the Internet.

Some properties of MDFS:

- All the files written to the file system will replicate over a number of the available data nodes creating data redundancy making data tolerant to hardware failure much greater.
- By building MDFS on the user-level in Java we enable the Data Nods to run any kernel level file system that i wants and there by making the Data Nodes none-dependent on both file system, platform and OS.
- By only building an API we do not limit the MDFS to a specific client and it can instead be used in many different settings and purposes. People will therefore be able to write any client software they want to utilize MDFS storage.

Document conventions

Mumfie = Name of the project

DFS = Distributed File System.

HDFS = Hadoop Distributed File system.

MDFS = Mumfie Distributed File system.

Node = A computer in a network of other computers.

TTL = Time To Live.

Data Node = A computer/node that stores raw data in MDFS .

Name Node = A coputer/node that stores and handels Meta-data for the MDFS.

OS = Operating System.

API = Application Programing Interface.

CLI = Command Line Interface.

must-have = features or functions that a software must-have.

nice-to-have = features or functions that would be nice to have in case of time software.

SSL = Secure Socket Layer, a way for encrypted data transfer over the TCP/IP protocol.

FUSE = A small unix opensorce software that enables third party user-level software to mount on the kernel level.

System description

MDFS is based on three main components which are interacting to create a user-level distributed file system, Name Node, Data Node and Client API. Outside of this we will have some other supporting software such as SQL-databases.

The over arching concept of MDFS is that all data is stored anonymously with just a uniq file name to identify them on the Data Nodes while the Name Node stores all meta-data for each files stored on all the Data Nodes. All requests regarding a file operation or information form a client are sent to the Name Node for processing and directions or information is returned to the client.request a file operation or information information.

A general description of MDFS is shown in figure below.

User interface

In MDFS there will be no User Interface. This is due to the fact that MDFS is not suppose to interact with a user directly but instead communicate with other softwares via a API.

A small CLI, GUI or automated software utilizing the simple functions of the API could be a nice addition but will be a considered a nice-to-have feature in the bounders of this project

System Functions

Req. No.	Requirement	Requirement Level
1	The Client API will be a small software that communicates with the Name and Data nodes and enables third part software to utilize generic file system functions over the TCP/IP-protocol	Must-Have
2	The Client API shall write and retrieve file and general meta-data from the Name Node	Must-Have
3	The Client API write and read raw data and appends necessary meta-data to and from the Data Nodes	Must-Have
4	The Name Node is a server software tha shall communicating with the Client and Data Nods over the TCP/IP-protocol and are providing structure in data storage.	Must-Have
5	The Name Node shall store all meta-data regarding files in MDFS in RAM	Must-Have
6	The Name Node shall replicates all meta-data from RAM in to a SQL database for safe storage continuously.	Must-Have
7	The Name Node shall provides client authentication.	Must-Have
8	The Name Node shall provides the client with necessary meta-data for reading and writing data to Data Nodes.	Must-Have
9	The Name Node shall track and communicates with all the Data Nodes.	Must-Have
10	The Data Node is a server software communicating with the Client and Name Node over the TCP/IP-protocol and are used for storage of raw data.	Must-Have
11	The Data Node shall retrieve files from the Client and store it on the local file system.	Must-Have

12	The Data Node shall replicates new files to a predetermined number of Data Nodes	Must-Have
13	The Data Node shall updates the Name Node with meta-data updates regarding storage of files.	Must-Have
14	Encryption for files transferred should be supported.	Nice-to-Have
15	SSL communication should be supported.	Nice-to-Have
16	Automated Building Latency graph for determining distance between different Nodes should be implemented.	Nice-to-Have
17	Enable support for multiple Name Nodes ensuring redundancy should be implemented	Nice-to-Have
18	Secure way of authentication of Data Nodes being trusted(not allowing every one to connect as a Data Node) should be implemented.	Nice-to-Have
19	Enable support for re-replication of file from a Data Node that is down should be implemented.	Nice-to-Have
20	Sample clients using the Client API. (CLI, GUI, Dropbox-Like-Client, FUSE-implementation) should be implemented.	Nice-to-Have

Non functional requirements

As MDFS is quite a extensive project with just the must-have system functional requirements and there for must-have non functional requirements are left outside this project.

Nice-To-Have

- Communication security, preventing eavesdropping on information and data transfers.
- Preventive masseurs for external attacks on a MDFS.

Storage of permanent data

- In regards to MDFS being a user-level file system we will store raw data on Data Nodes local file system as regular data.
- Meta-data from the Name Node will be stored in a SQL-database.

Description of limitations

Considering this is a project that will be used outside of the university there is no real limitations but time. There will be limitations, but since it is hard to estimate some features we will leave this open for now. It seems likely to be the nice-to-have features that will be left out of this project but if time permits we will try to make as many of them as possible. In general security features regarding trust between Nodes and Client are most likely not going to be covered in this project.

Use Case Description

Our imaginary person, Emil K,has a good experience with unix operating systems and software. Emil K writes a quick bridge between FUSE and the MDFS Clients API making MDFS mountable. He decides to store this file to try the system. For Emil K this looks nothing like a complex software but just another file system mounted on his computer. He decides to look into the documentation to see that this system to how it works behind the scenes.

Emil K finds out that it works something like this:

1. When FUSE is being mounted Emil K have to provide user credentials.
2. When FUSE writes the file through the Client API:
    a. A request for writing is sent to the Name Node
    b. The Name Node returns instruction on what Data Node to write
    c. The Client sends the file data to the Data Node for permanent storage
    d. The Data Node writes the data to the local file system and replicates it to other Data Nodes
3. When FUSE reads a file through the Client API:
    a. A request for the file is sent to the Name Node
    b. The Name Node returns meta-data regarding the file and what data nodes that holds the file
    c. The Client API sends a request for the file to one of the listed Data Nodes
        i. If the data node fails to respond, the Client tries another of the listed Data Nodes
    d. The Data Node retrieves the file from the local file system and returns it to the Client
4. When FUSE accesses meta-data through the Client API:
    a. A request for our file system meta-data is sent to the Name Node
    b. The Name Node looks up the request in its meta-data tree and returns it to the Client