HCFS

Filesystem Compatibility with Apache Hadoop

See this link for Community Progress and Participation on these topics

Apache Hadoop is built on a distributed filesystem, HDFS, *Hadoop Distributed File System*, capable of storing tens of Petabytes of data. This filesystem is designed to work with Apache Hadoop from the ground up, with location aware block placement, integration with the Hadoop tools and both explicit and implicit testing.

Apache Hadoop also works with other filesystems, the platform specific "local" filesystem, Blobstores such as Amazon S3 and Azure storage, as well as alternative distributed filesystems.

All such filesystems (including HDFS) must link up to Hadoop in two ways.

1. The filesystem looks like a "native" filesystem, and is accessed as a local FS, perhaps with some filesystem-specific means of telling the MapReduce layer which TaskTracker is closest to the data.
2. The filesystem provides an implementation of the `org.apache.hadoop.fs.FileSystem` class (and in Hadoop v2, in implementation of the `FileContext` class}

Implementing the `FileSystem` class ensures that there is an API for applications such as MapReduce, Apache HBase, Apache Giraph and others can use -including third-party applications as well as code running in a MapReduce job that wishes to read or write data.

The selection of which filesystem to use comes from the URI scheme used to refer to it -the prefix `hdfs:` on any file path means that it refers to an HDFS filesystem; `file:` to the local filesystem, `s3:` to Amazon S3, `ftp:` FTP, `swift:` OpenStackSwift, ...etc.

There are other filesystems that provide explicit integration with Hadoop through the relevant Java JAR files, native binaries and configuration parameters needed to add a new schema to Hadoop, such as `fat32:`

All providers of filesystem plugins do their utmost to make their filesystems are compatible with Apache Hadoop. Ambiguities in the Hadoop APIs do not help here -as a lot of the expectations of Hadoop applications are set not by the `FileSystem` API, but the behavior of HDFS itself -which makes it harder to distinguish "bug" from "feature" in the behavior of HDFS.

The Hadoop developers are (as of April 2013), attempting to define the Semantics of the Hadoop FileSystem more rigorously as well as adding better test coverage for the filesystem APIs. This will ensure that we can keep the filesystem implementations that ship with Hadoop ~~HDFS itself, and those classes that connect to other filesystems, currently `s3:`, `s3n:`, `file:`, `ftp:`, `webhdfs`~~ consistent with each other, and compatible with existing applications.

This formalisation of the API will also benefit anyone who wishes to to provide a library that lets Hadoop applications work with their FileSystem -such people have been very constructive in helping define the FileSystem APIs more rigorously.

The list below contains links to information about some of the additional FileSystems and Object Stores for Apache Hadoop

Known contributors actively commited to working on HCFS initiative... (Add your organization here) !

- HDFS
- GlusterFS
- OrangeFS
- SwiftFS
- GridGain

Other known members of the HCFS community:

- /* Alphabetical order, no endorsements, please */. */
- Windows Azure Blob Storage
- CassandraFS
- CephFS
- CleverSafe Object Store
- Google Cloud Storage Connector
- Lustre
- MapR FileSystem
- Quantcast File System
- Symtantec Veritas Cluster File System

Even if the filesystem is supported by a library for tight integration with Apache Hadoop, it may behave differently from what Hadoop and applications expect: this is something to explore with the supplier of the filesystem.

The Apache Software Foundation can make no assertion that a third party filesystem is *compatible* with Apache Hadoop: these are claims by the vendors which the Apache Software Foundation do not validate.

What the ASF can do is warn that our own BlobStore filesystems (currently `s3:`, `s3n:` and `swift:`) are not complete replacements for `hdfs:`, as operations such as `rename()` are only emulated through copying then deleting all operations, and so a directory rename is not atomic -a requirement of POSIX filesystems which some applications (MapReduce) currently depend on.

Similarly the local `file:` filesystem behaves different only different operating systems, especially regarding filename case and whether or not you can delete open files. If your intent is to write code that only ever works with the local filesystem, always test on the target platform.