HDFS NFS Gateway

{brandon, suresh}@hortonworks.com

April 24, 2013

Introduction

Access HDFS is usually done through HDFS API or its web API. Lack of seamless integration with client's file system makes it difficult for users and impossible for some applications to access HDFS. NFS interface support is one way for HDFS to support such easy integration. With this, HDFS can be accessed using HDFS client, web API and the NFS protocol. HDFS will be easier to access and be able support more applications and use cases.

This document describes the NFS gateway for accessing HDFS. "Hadoop+FUSE" could be used to provide an NFS interface for HDFS. However it has many known problems and limitations. Therefore, it's prefered to have the native NFS interface support to HDFS.

User Scenarios

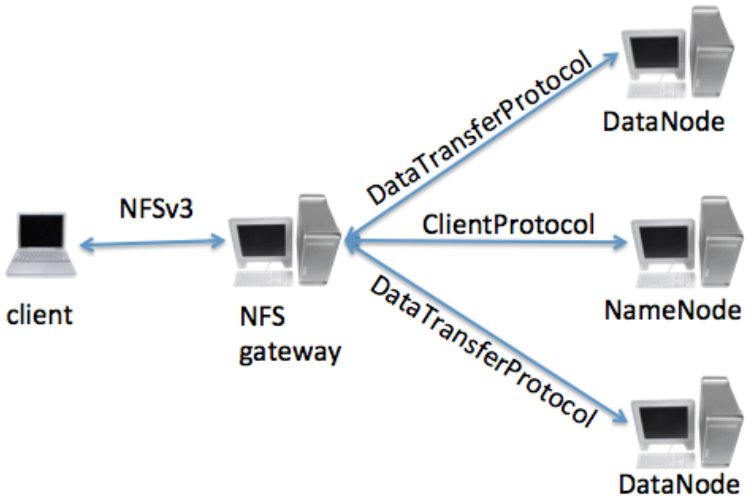
1. **File browsing and downloading:** Users and applications can browse the files saved on HDFS and download from HDFS.
2. **File uploading:** Users and applications can upload files on HDFS.
3. **Data streaming**: Applications can stream data directly to HDFS.

High Level Requirements

1. NFS version 3: simply because V3 is still the most widely used NFS protocol. It's well understood and easy to implement.
2. Single export per namespace. HDFS root ('/') is the single export.
3. NFS gateway must be stateless to simplify future addition of HA capability.

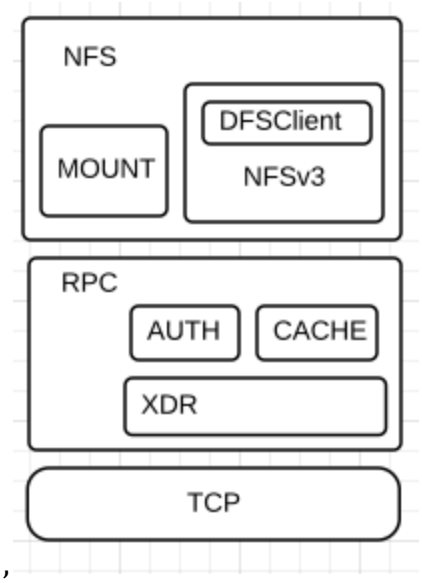
Design and Implementation

NFS gateway's main functionality is the translation between NFS protocol and HDFS access protocols as shown in the following diagram.



NFS gateway is a service in user space. Being in user space gives more implementation flexibility and makes it easier to integrate with other Hadoop building blocks.

The following diagram shows the internal and related modules of the NFS gateway.



,

- TCP: As it's costly to handle resend write requests. The first phase will only support NFS over TCP since it's more reliable comparing with UDP.
- XDR: External Data Representation, provides a standard way of representing a set of data types on a network. See RFC 4506.
- ONCRPC: Open Network Computing Remote Procedure Call. See - RFC 1831.
- AUTH: ONC RPC has several authentication flavors. NFS gateway will support AUTH_SYS in the

first phase. NFS clients use UID/GID to communicate with server, while HDFS uses names for user and group. This component does conversion between the id and name.

- CACHE: Duplicate Request cache, a way to help a server give correct responses to certain types of replayed operations. Note that, some idempotent NFS requests(e.g., write) for traditional NFS servers are not idempotent any more for the NFS gateway due to the HDFS readonly nature.
- NFSv3: It handles all the NFS requests and uses DFSClient (or FileSystem) objects to access HDFS.
- MOUNT: It handles all the MOUNT requests. Unlike mountd on Unix/Linux systems, it doesn't export nodes of host namespace. It only exports part of the HDFS namespace.

Implementation Challenges

Given the current HDFS design and architecture, there are a few technical challenges to support this NFS interface. Here are some of them and their initial solutions.

- HDFS is a read only file system with append. NFSv3 is stateless and has no open/close call. The question is when to close an HDFS file. The solution is to close the stream after it's idle(no write) for a certain period(e.g., 10 seconds). The subsequent write will become append and open the stream again.
- Due to the nature of many NFS client implementations, Kernel may reorder sequential writes before sending to the NFS server. In this case, sequential writes could arrive at the server at random order. According to reference [1], waiting a short period(e.g., 10 milliseconds) could collect most of the reordered writes. Therefore, a write is considered as random write when its prerequisite writes can't arrive in a certain period.
- NFSv3 uses a file handle to access files and the file handle is unique cross the lifetime of the exported file server. However HDFS access is file path based. This problem will be resolved by HDFS-4489, which introduces inode id concept into HDFS. The NFS file handle will eventually include namespaceId, snapshotId and inodeId.
- By default, the DFSClient inside NFS gateway doesn't flush data after each write, and thus the user may not be able to read back the written data immediately. The solution is to read the data from the buffer in NFS gateway instead of DataNode.

Organizing the work in terms of jiras

This jira requires work to be done both in Hadoop common and HDFS. The following will be done in common:

- ONC RPC related functionality will be done under `hadoop-common-project/hadoop-oncrpc`
- NFS3 related functionality will be done under `hadoop-common-project/hadoop-nfs3`

HDFS specific work based on the above common functionality that acts as gateway from NFS3 to HDFS will be done under `hadoop-hdfs-project/hadoop-hdfs-nfs`.

Future work

The first phase of implementation will focus on getting an early version of NFS gateway that works correctly and reliably in place. The following work will be done in subsequent phases:

- HA
- Other NFS protocol versions
- Better security, e.g., Kerberos
- Submount support

Reference:

[1] http://www.eecs.harvard.edu/sos/techs/tr-06-02.ps