Distributed Systems

Project Part 1

HDFS

Deadline : 23rd March 2016

This part of the project requires to implement a distributed file system similar to HDFS. The distributed files system will have two major components NameNode and DataNode. Functionality of these components is as follows:

- **NameNode :**
  It supports all the file operations like open, close, list. The file system supports a block mechanism similar to HDFS. So, NameNode also takes care of managing block allocations and get locations of existing blocks. It will also handle the communication with DataNode.

- **DataNode :**
  Used for reading and writing blocks.

- **Client :**
  - Command line interface to interact with the NameNode.
  - Possible commands given are :
    - get (Writes a file to local filesystem from HDFS)
    - put (Writes a file to HDFS from local file system)
    - list (Displays all files present in HDFS)
  - Handles reads and writes from DataNode on receiving metadata information from NameNode about a file.

- **Important Points:**
  - Use RMI (Java) or Sun RPC (C/C++) for communication
  - Server should be multi-threaded
  - No update operation
  - 2-way cascading replication
  - You can have a configuration file of your own
  - Add CRC for communication (optional but recommended)
  - All the parameters and return types are byte arrays (need to marshal and unmarshal actual objects)

- **Detailed Workflows:**
    - All APIs take a byte[] as an input param, and return a byte[] as an output. The input byte[] is a Google protobuf. Similarly, the output byte[] is a Google protobuf.

    - **Open**
        - Assign initial block.
        - Unique handle for each opened file.
    - **Write**
        - Write contents to the assigned block number

        Flow for writing a file:

```
openFile("filename");
In a loop, do the following:
    Call assignBlock() using handle from openFile
    Obtain a reference to the Remote DataNode object using the first entry in the DataNodeLocation
    Call writeBlock() to the DataNode
closeFile()
```

    - **Read**
        - Get all block locations for the file
        - Read blocks in sequence

        Flow for reading a file:

```
openFile("filename");
In a loop, do the following for each block:
    getBlockLocations() using handle from openFile
    Obtain a reference to the Remote DataNode object using the a random entry in the DataNodeLocation
    Issue a readBlock() to the DataNode
closeFile()
```

    - **Close**
        - Close the file after read or write request.

    - **No update operation**

    - All responses from servers are with a status variable denoting the failure or success of the request.

    - Heartbeat should be executed periodically to know the status of DataNode BlockReport.

    - DN sends blockReport() to inform the NN about all the blocks it has.

    - For each file, NN knows just the filename and list of blocks. NN does not store the locations. Once DNs send blockReports, NN knows the locations of each block and tracks this information in memory - it never writes this information to a disk. NN has to maintain the list of files, and the blocks in each file in a directory (or a file) in the local OS.

    - Clients and DN discover the NN from a conf file and read from a standard location. The conf file contains the ip/port of the NN.

Possible Cluster Setup Configurations

- Each of the two members can have one virtual machine installed by using VMware or other virtualization tools. You'll then have four machines for the HDFS cluster (Two VMs and two host machines). Connecting the machines with the LAN wire with static IPs and configuring VMs in bridged mode in VMware should establish all working connections to resume actual work.
- Other than VMs, you can also use docker containers having setup for Java or C++.
- Don't expect support in establishing network and working configuration.