In this file, we will briefly introduce how to deploy and run the FastPR prototype. More design details can be referred to our paper "Fast Predictive Repair in Erasure-Coded Storage" which appears at the 49th IEEE/IFIP International Conference on Dependable Systems and Networks (IEEE/IFIP DSN'19). If you have any question about the deployment, please feel free to contact me at zhirong.shen2601@gmail.com

1. Preparation

1.1. Installing necessary packages

- make and g++

```
$ sudo apt-get install make g++
```

- ISA-L (needed in Hadoop 3.1.1)

```
$ git clone https://github.com/01org/isa-l.git
$ cd isa-l/
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```

- Hadoop 3.1.1

```
$ wget http://apache.01link.hk/hadoop/common/hadoop-3.1.1/hadoop-3.1.1-src.tar.gz
$ tar -zxvf hadoop-3.1.1-src.tar.gz
$ cd hadoop-3.1.1-src/
$ mvn package -DskipTests -Dtar -Dmaven.javadoc.skip=true -Drequire.isal -Pdist,native
-DskipShade -e     # use maven to compile the HDFS for supporting erasure coding
```

1.2. Hadoop Configurations

- Set the environment variables for HDFS and JAVA in ~/.bashrc. The following is an sample used in our testbed.

```
export JAVA_HOME=/home/ncsgroup/java
export HADOOP_HOME=/home/ncsgroup/zrshen/hadoop-3.1.1
export PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH
```

- Make the environment variables work

```
$ source ~/.bashrc
```

- Configure the configuration files under the folder hadoop-3.1.1/etc/hadoop/, including core-site.xml, hadoop-env.sh, hdfs-site.xml, and workers. The following example provides a reference for the Hadoop configuration. Section 1.3 shows an example to configure Hadoop 3.1.1 in a cluster.

1.3 Example Architecture

The following table shows an example architecture used in our testbed. The Coordinator in FastPR is collocated with the NameNode of Hadoop 3.1.1 and the Agent in FastPR is run on the DataNode of Hadoop 3.1.1.

IP Address	Roles in Hadoop	Roles in FastPR
192.168.10.51	NameNode	Coordinator
192.168.10.52	DataNode	Agent
192.168.10.53	DataNode	Agent
192.168.10.54	DataNode	Agent
192.168.10.55	DataNode	Agent
192.168.10.56	DataNode	Agent
192.168.10.57	DataNode	Agent
192.168.10.58	DataNode	Agent
192.168.10.59	DataNode	Agent
192.168.10.110	Client	------

We then configure the four configuration files under the folder of $HADOOP_HOME/etc/hadoop/. We show the configurations in our testbed as follows.

- core-site.xml

```
<configuration>
<property><name>fs.defaultFS</name><value>hdfs://192.168.10.51:9000</value></property>
<property><name>hadoop.tmp.dir</name><value>/home/ncsgroup/zrshen/hadoop-3.1.1</value>
</property>
</configuration>
```

- hadoop-env.sh

```
export JAVA_HOME=/home/ncsgroup/java
export HADOOP_HOME=/home/ncsgroup/zrshen/hadoop-3.1.1
export HDFS_NAMENODE_USER=ncsgroup        # replace "ncsgroup" with your username
export HDFS_DATANODE_USER=ncsgroup
export HDFS_SECONDARYNAMENODE_USER=ncsgroup
export YARN_RESOURCEMANAGER_USER=ncsgroup
export YARN_NODEMANAGER_USER=ncsgroup
```

- hdfs-site.xml

```
  <configuration>
  <property><name>dfs.replication</name><value>1</value></property>
  <property><name>dfs.blocksize</name><value>67108864</value></property>
  </configuration>
```

- workers

```
192.168.10.52
192.168.10.53
192.168.10.54
192.168.10.55
192.168.10.56
192.168.10.57
192.168.10.58
192.168.10.59
```

1.4 Write Erasure-Coded Data to HDFS

- At the hadoop client side, create a file named "testfile", whose size should be multiple times of the data size of a stripe (i.e., the size of data chunks in a stripe). As we use RS(5,3) as an example in the following descriptions, we can create a file with the size of 30GB.

```
dd if=/dev/urandom of=testfile bs=1M count=30720    # create a random file (30GB)
```

- At the client side, select and enable an erasure coding scheme and write data to the HDFS. Here we use RS(5,3) as an instance. If you use other erasure coding schemes, please ensure that the size of testfile (in the last step) should be multiple times of the data size of a stripe.

```
$ hadoop fs -mkdir /ec_test                           # create a folder named /ec_test
$ hdfs ec -listPolicies                               # list the policies supported by HDFS
$ hdfs ec -enablePlicy -policy RS-3-2-1024k           # enable an erasure coding policy
$ hdfs ec -setPolicy -path /ec_test -policy RS-3-2-1024k  # set ec policy to /ec_test
$ hdfs ec -getPolicy -path /ec_test                   # confirm the ec policy of /ec_test
$ hadoop fs -put testfile /ec_test                    # write testfile to /ec_test
```

2. Configuration

2.1. Introduction to The Configurations

The configuration of FastPR is realized by a XML file named config.xml, which is stored under the folder "metadata". The config.xml specifies the following parameters and their physical meanings.

Parameters	Physical meanings
erasure_code_k	Number of data chunks in a stripe
erasure_code_n	Number of data and parity chunks in a stripe
peer_node_num	Number of nodes in a system
packet_size	Size of packet in read, transmission, and write (to enable pipelining)
chunk_size	Size of a chunk, also called block, in HDFS (in unit of bytes)
meta_size	Size of metadata of a chunk in HDFS (in unit of bytes)
stripe_num	Number of stripes to be repaired
disk_bandwidth	Measured disk bandwidth capacity (in unit of MB/s)
network_bandwidth	Measured network bandwidth capacity (in unit of Gb/s)
Coordinator_ip	IP address of the NameNode
repair_scenario	"scatteredRepair" OR “hotStandbyRepair”
hotstandby_node_num	Number of hot-standby nodes
peer_node_ips	IP addresses of peer nodes
hotstandby_node_ips	IP addresses of hot-standby nodes
local_ip	IP address of this node
local_data_path	Absolute path that stores the HDFS data chunks (also called blocks)

2.2. Configuration Example

The following table shows the configuration parameters used in our testbed.

```
<setting>
<attribute><name>erasure_code_k</name><value>3</value></attribute>
<attribute><name>erasure_code_n</name><value>5</value></attribute>
<attribute><name>peer_node_num</name><value>8</value></attribute>
<attribute><name>packet_size</name><value>4194304</value></attribute>
<attribute><name>chunk_size</name><value>67108864</value></attribute>
<attribute><name>meta_size</name><value>524295</value></attribute>
```

```
<attribute><name>stripe_num</name><value>1000</value></attribute>
<attribute><name>disk_bandwidth</name><value>100</value></attribute>
<attribute><name>network_bandwidth</name><value>1</value></attribute>
<attribute><name>coordinator_ip</name><value>192.168.10.51</value></attribute>
<attribute><name>repair_scenario</name><value>scatteredRepair</value></attribute>
<attribute><name>hotstandby_node_num</name><value>3</value></attribute>
<attribute><name>peer_node_ips</name>
<value>192.168.10.52</value>
<value>192.168.10.53</value>
<value>192.168.10.54</value>
<value>192.168.10.55</value>
<value>192.168.10.56</value>
<value>192.168.10.57</value>
<value>192.168.10.58</value>
<value>192.168.10.59</value>
</attribute>
<attribute><name>hotstandby_node_ips</name>
<value>192.168.10.60</value>
<value>192.168.10.61</value>
<value>192.168.10.62</value>
</attribute>
<attribute><name>local_ip</name><value>192.168.10.51</value></attribute>
<attribute><name>local_data_path</name><value>/home/ncsgroup/zrshen/hadoop-
3.1.1/dfs/data/current/BP-679808751-192.168.10.51-
1551859882159/current/finalized/subdir0</value></attribute>
</setting>
```

3. Deployment of FastPR

3.1 Download and Install

We can deploy the FastPR as follows:

- If you choose to download the source code from the project website of FastPR, then use the following commands to extract the files of FastPR and compile FastPR.

  ```
  $ tar -zxvf fastpr-v1.0.tar.gz
  $ cd fastpr-v1.0/Jerasure && make  # compile the Jerasure library needed in FastPR
  $ cd .. && make                    # compile FastPR
  ```

- If you choose to download the source code from GitHub, then use the following commands to compile FastPR

  ```
  $ git clone https://github.com/shenzr/fastpr.git
  $ cd fastpr/Jerasure && make
  $ cd .. && make
  ```

  The above commands will generate three executable files, named "**FastPRCoordinator**", "**FastPRPeerNode**", and "**FastPRHotStandby**". The roles of these three files are as follows.

Executable files	Functionality
FastPRCoordinator	Coordinator in our paper. It will determine which chunks to be reconstructed and migrated, and issue commands to guide the repair.
FastPRPeerNode	It runs as an agent in our paper. It receives commands from the Coordinator, parses the commands to understand its role (i.e., sender or receiver), and does the jobs (e.g., read which data chunk and send it to which DataNode, or receive how many data chunks and the name of the repaired chunk) specified in the commands.
FastPRHotStandby	It also runs as an agent in our paper. It will dedicatedly receive data for repair.

- Fill in the information (including system information, and the information of erasure coding) in the metadata/config.xml (see our example in 2.2)

3.2 Run FastPR

- Start the agents at the DataNode. For scattered repair, start "FastPRPeerNode" at the DataNodes.

```
$ ./FastPRPeerNode
```

- For hot-standby repair, start "FastPRPeerNode" at the DataNode of the original system and start "FastPRHotStandby" at the DataNode that serves as the hot-standby node

```
$ ./FastPRPeerNode              # for DataNode of the original system
$ ./FastPRHotStandby            # for DataNode to serve as the hot-standby node
```

- Run the FastPRCoordinator at the NameNode. The command format is

```
./FastPRCoordinator (id_of_STF_node) (repair_method) (num_of_repair_chunks)
```

3.3 An Example

For example, the following command is to repair the 50 chunks of the STF node (suppose node id is 0) using migration

```
$ ./FastPRCoordinator 0 migration 50   # for migration
```

The following command is to repair the 50 chunks of the STF node (suppose node id is 0) using random repair

```
$ ./FastPRCoordinator 0 random 50      # for random repair
```

The following command is to repair the 50 chunks of the STF node (suppose node id is 0) using FastPR

```
$ ./FastPRCoordinator 0 fastpr 50      # for FastPR
```

When the program runs, the coordinator will find a DataNode to store each repaired data chunk and its metadata chunk. The repaired data chunks and their metadata chunks are stored at the path "$local_data_path/subdir0" of the DataNode ($local_data_path is specified in the config.xml file (see Section 2.1)).