ECPipe User Guide

ADSLab @ CUHK

Release: June 2017

Contents

1 Installation

1.1 System Requirement

We have tested ECPipe on Ubuntu 14.04 LTS.

1.2 Software Requirement

Some of the packages can be downloaded from the project website
http://adslab.cse.cuhk.edu.hk/software/ecpipe.

- g++ v4.8.4

  We need a C++ compiler that supports the C++11 standard.

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

- Redis v3.2.8

  Download **redis-3.2.8.tar.gz** and install it.

  ```
  $ tar -zxvf redis-3.2.8.tar.gz
  $ cd redis-3.2.8
  $ make
  $ sudo make install
  ```

  Install redis as a background daemon. You can just use the default settings.

  ```
  $ cd utils
  $ sudo ./install_server.sh
  ```

  Configure redis to be remotely accessible.

  ```
  $ sudo /etc/init.d/redis_6379.sh stop
  ```

  Edit */etc/redis/6379.conf*. Find the line with bind 127.0.0.0 and modify it to bind 0.0.0.0, then start redis.

  ```
  $ sudo /etc/init.d/redis_6379.sh start
  ```

- hiredis

  Download **hiredis.tar.gz** and install it.

```
$ tar -zxvf hiredis.tar.gz
$ cd hiredis
$ make
$ sudo make install
```

- gf-complete

  Download **gf-complete.tar.gz** and install it (note that you may need to install autoconf and libtool first).

```
$ tar -zxvf gf-complete.tar.gz
$ cd gf-complete
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```

1.3 Compile ECPipe

Download **ECPipe-1.0.tar.gz** and compile the source code.

```
$ tar -zxvf ECPipe-1.0.tar.gz
$ cd ECPipe
$ make
```

In the following, we use `~ecpipe/` to refer to the `ECPipe/` directory or the working directory where ECPipe is installed.

2 Standalone Test of ECPipe

We can test ECPipe as a standalone system without being integrated to any existing distributed file system. To demonstrate how ECPipe works, we consider an example in which there is a cluster of five nodes. One node is the coordinator, and the other four nodes are helpers (clearly, you can have any number of four helpers). We assume that ECPipe has been installed in all these five nodes. Table 1 shows the information of the example cluster.

Note: The source code uses `packets` to represent slices in the paper.

Hostname	IP address	Type
node1	192.168.0.1	coordinator
node2	192.168.0.2	helper
node3	192.168.0.3	helper
node4	192.168.0.4	helper
node5	192.168.0.5	helper

Table 1: Details of the example cluster.

2.1 Prerequisites

2.1.1 Configuration File

We configure the settings of ECPipe via the configuration file `config.xml` in XML format. Table 2 explains the meaning of each property. We provide a sample configuration file `~ecpipe/conf/config.example.xml`. You can copy this file as `~ecpipe/conf/config.xml` to each node and modify it with proper settings.

Property name	**Description**
`erasure.code.k`	Parameter k in erasure coding.
`erasure.code.n`	Parameter n in erasure coding.
`rs.code.config.file`	Full path to the coding matrix file (see Section 2.1.2).
`packet.size`	Size of a packet (called slice in our paper) in units of bytes.
`packet.count`	Number of packets in a block.
`degraded.read.policy`	Three repair policies supported: `conv` (i.e., conventional repair), `ppr` (i.e., the PPR approach), and `ecpipe` (i.e., our ECPipe approach).
`ecpipe.policy`	The repair policy if ECPipe is used, either `basic` or `cyclic`.
`coordinator.address`	IP address of the coordinator (e.g., 192.168.0.1 in our example)
`file.system.type`	Three types are supported: `standalone`, `HDFS`, or `QFS`.
`stripe.store`	Full path to the directory where the stripe metadata is stored.
`block.directory`	Full path to the directory where the coded blocks are stored.

`helpers.address`	A list of IP addresses of all helpers, in the form of `zone/IP address`, where `zone` denotes the zone (e.g., rack or data center). For example, suppose that the helpers in our example cluster are all in zone `default`. Then we configure the following:
	`default/192.168.0.2`
	`default/192.168.0.3`
	`default/192.168.0.4`
	`default/192.168.0.5`

`local.ip.address`	IP address of the node itself (e.g., `192.168.0.1` for node1)
`path.selection.enabled`	`true` if weighted path selection is enabled; `false` otherwise.
`link.weight.config.file`	Full path to the weight matrix file.

Table 2: Details of the configuration file `config.xml`.

2.1.2 Coding Matrix File

The coding matrix file describes an $(n-k) \times k$ coding matrix that specifies the coefficients for generating $n-k$ coded blocks from k uncoded blocks. For example, our example cluster uses the file `~ecpipe/conf/rsEncMat_3_4` to construct a simple coding matrix for $(n,k) = (4,3)$:

$$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}.$$

The file `~ecpipe/conf/rsEncMat_6_9` describes a more complicated coding matrix for $(n,k) = (9,6)$:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 225 & 151 & 172 & 82 & 200 \\ 1 & 123 & 245 & 143 & 244 & 142 \end{bmatrix}.$$

2.1.3 Create Erasure-Coded Blocks

Before we start our standalone test, we first need to create a stripe of erasure-coded blocks. We have provided a program for the block generation under `~ecpipe/test/`.

```
$ cd ~ecpipe/test
$ make
$ dd if=/dev/urandom of=input.txt bs=1M count=3
$ ./createdata ../conf/rsEncMat_3_4 ./input.txt 3 4
```

Note: You can just run the command `./createdata` to get the usage of this program.

Suppose that $(n,k) = (4,3)$. In `~ecpipe/test`, we create three files of uncoded blocks `file_k1`, `file_k2`, and `file_k3`, and one file of coded block `file_m1`.

We can distribute the four blocks across the four helpers, each of which stores one block under the path specified by `block.directory`.

The coordinator also stores the stripe metadata under the path specified by `stripe.store`. In our example, the coordinator has four files: `rs:file_k1_1001`, `rs:file_k2_1001`, `rs:file_k3_1001`, and `rs:file_m1_1002`, all of which have the following content:

```
file_m1_1002:file_k1_1001:file_k2_1001:file_k3_1001
```

The file name `rs:file_k1_1001` means that the block uses Reed-Solomon (RS) codes and the block name is `file_k1`. The tail 1001 (resp. 1002) means the block is an uncoded block (resp. coded block).

2.2 Start ECPipe

The start script is in `~ecpipe/script`.

```
$ python scripts/start.py
```

2.3 Degraded Read Test

Use ssh to connect to any one of the helpers (not the coordinator). For example, we can issue a degraded read at the helper that stores `file_k1`.

```
$ ./ECPipeClient file_k1
```

This command repairs `file_k1` from other helpers (i.e., a degraded read). You can measure the time for the degraded read. The output file is `testfileOut`, which should be identical to the original block `file_k1`.

2.4 Stop ECPipe

The stop script is in `~ecpipe/script`.

```
$ python scripts/stop.py
```

2.5 Weighted Path Selection

ECPipe supports weighted path selection by setting `path.selection.enabled` to be `true` and specifying the weight matrix file through `link.weight.config.file`. Note that we currently only support `degraded.read.policy` = `ecpipe` and `ecpipe.policy` = `basic`.

The weight matrix file contains an $N \times N$ matrix, where N is the total number of helpers in ECPipe cluster (e.g., $N = 4$ in our example). Let W denote the matrix. The element $W[i][j]$ denotes the link weight from the i-th helper to the j-th helper. Note that $W[i][j]$ may be different from $W[j][i]$. We provided an sample weight matrix file `~ecpipe/conf/linkWeightMat` for $N = 4$.

In deployment, you can first measure the network performance between every pair of helpers and configure the values of the matrix.

3 Hadoop-20 Integration

In this section, we explain how we integrate ECPipe into Hadoop-20.

3.1 Prerequisites

The following packages need to be first installed.

- ant

  Download **apache-ant-1.9.9-bin.tar.gz**.

  ```
  $ tar -zxvf apache-ant-1.9.9.-bin.tar.gz
  ```

  Set the environment variables `ANT_HOME` and `PATH` (you may need to set `ANT_OPTS` if you are behind a proxy.)

  ```
  export ANT_HOME=~/apache-ant-1.9.9
  export PATH=$PATH:$ANT_HOME/bin
  ```

- java8

  ```
  $ sudo add-apt-repository ppa:webupd8team/java
  $ sudo apt-get update
  $ sudo apt-get install oracle-java8-installer
  $ sudo apt-get install oracle-java8-set-default
  ```

  Set the environment variable `JAVA_HOME`.

  ```
  export JAVA_HOME=/usr/lib/jvm/java-8-oracle
  ```

- zlib

  ```
  $ sudo apt-get install zlib1g-dev
  ```

3.2 Install Hadoop-20

Download **hadoop-20.tar.gz** (we provided a copy on our project website).

```
$ tar -zxvf hadoop-20.tar.gz
```

We use `~hadoop-20` to refer to the directory where Hadoop-20 is installed.

```
export PATH=$PATH:~hadoop-20/bin
```

Edit `~ecpipe/hadoop-20-integrate/install.sh` with the proper directory names `~ecpipe` and `~hadoop-20`. Then execute the script.

```
$ ./install.sh
```

3.3 Cluster Setting

To demonstrate how ECPipe works with hadoop-20, we consider an example in which there is a cluster of six nodes. We assume that ECPipe has been installed in all these 6 nodes. Hadoop-20 has been installed in all helper nodes.

Node	IP	Hadoop-20 Node Type	ECPipe Node Type
node1	192.168.0.1	not applicable	coordinator
node2	192.168.0.2	NameNode, RaidNode	helper
node3	192.168.0.3	DataNode	helper
node4	192.168.0.4	DataNode	helper
node5	192.168.0.5	DataNode	helper
node6	192.168.0.6	DataNode	helper

Note: Make sure that the NameNode can ssh to any DataNode via public key authentication without password. The public/private keys can be set up via `ssh-keygen`.

3.4 Hadoop Configuration

Hadoop configuration files are in XML format. We provide sample configuration files in `~ecpipe/hadoop-20-integrate/conf`. You can copy them to `~hadoop-20/conf`.

- hadoop-env.sh

Property name	Description
`JAVA_HOME`	Your JAVA_HOME
`HADOOP_USERNAME`	Your linux user name

- core-site.sh

Property Name	Description
`fs.default.name`	IP address and port number (e.g., `hdfs://192.168.0.2:9000`)
`topology.script.file.name`	Full path to rackAware.sh
`hadoop.tmp.dir`	Full path to the hadoop-20 data directory

- hdfs-site.xml

  We only highlight the important properties that are related to our work, as listed in Table 3. For other properties, you may follow the default settings.

  Table 4 shows the properties of `raid.codecs.json`.

Property Name	**Description**
`ecpipe.coordinator`	IP address of the coordinator (e.g., 192.168.0.1 in our case).
`ecpipe.packetsize`	The slice size in our paper (e.g. `32768` in our example).
`ecpipe.packetcnt`	The number of slices (e.g. `32` in our example).
`ecpipe.local.addr`	IP address of the node itself.
`dfs.use.inline.checksum`	Assumed to be `false`.
`use.ecpipe`	Set to `true` to enable ECPipe.
`num.src.node`	Parameter k in erasure coding.
`num.parity.node`	Parameter $n - k$ in erasure coding.
`dfs.http.address`	Address to which NameNode web UI listens (e.g. `192.168.0.2:50070`).
`dfs.replication`	Replication parameter for uncoded blocks.
`hdfs.raid.parity.initial.repl`	Replication parameter for coded blocks.
`raid.config.file`	Full path to the RAID configuration file.
`hdfs.raid.local.stripe.dir`	Full path to the stripe store directory.
`dfs.block.size`	HDFS block size (in bytes).
`raid.codecs.json`	It is in JSON format. See Table 4.

Table 3: Details of hdfs-site.xml.

Object Name	**Description**
`id`	Identifier of the coding scheme.
`parity_dir`	Parity directory (e.g., `/parity`).
`stripe_length`	Parameter k in erasure coding.
`parity_length`	Parameter $n - k$ in erasure coding.
`erasure_code`	Class name of the erasure code implementation (see the sample XML file for configuration).
`dir_raid`	Set it to `true` for directory raid.

Table 4: Details of raid.codecs.json.

- raid.xml

  The default `srcPath` is `hdfs://192.168.0.2:9000/user/username/raidTest`. Set the proper `srcPath` according to your configuration.

- masters

  This file contains one line with your NameNode IP address.

```
192.168.0.2
```

- slaves

  This file contains multiple lines, each of which is a DataNode IP address.

```
192.168.0.3
192.168.0.4
192.168.0.5
192.168.0.6
```

3.5 Start Hadoop

- Format the Hadoop cluster.

```
$ hadoop namenode -format
```

- Start the Hadoop cluster.

```
$ start-dfs.sh
```

- Check whether the Hadoop cluster has started correctly.

```
$ hadoop dfsadmin -report | grep total
```

  If the result indicates that there are 4 DataNodes, then the Hadoop cluster starts correctly.

- Write data into HDFS.

  For example, we create a file of 3MB using `dd` and write it into HDFS.

```
$ cd ~/hadoop-20
$ dd if=/dev/urandom of=file.txt bs=1048576 count=3
$ hadoop dfs -put file.txt raidTest/input
```

- Start RaidNode for erasure coding.

```
$ start-raidnode.sh
```

- Check whether erasure-coded data is ready.

```
$ hadoop fsck / -files -blocks -locations
```

  We can see four blocks from the results.

- Stop RaidNode.

```
$ stop-raidnode.sh
```

3.6 Start ECPipe

- Identify the block directory. For example,

```
$ ssh node3
$ find -name "finalized"
```

  You can ssh to any DataNode to execute this command, and the result is the block directory.

- config.xml

  Please note that we only support `degraded.read.policy` = `ecpipe`, and `ecpipe.policy` = `basic` for now. Check `~ecpipe/conf/config.xml` to see if the following properties are properly set.

Property Name	**Description**
`file.system.type`	Set it to `HDFS` here.
`stripe.store`	Your HDFS stripe store directory.
`block.directory`	Your HDFS block directory.
`helpers.address`	List of helper addresses.

  Other configurations are similar to what we introduced earlier.

- Start ECPipe.

```
$ cd ~ecpipe
$ python scripts/start.py
```

3.7 Degraded Read Test

- Ssh to one of the Hadoop DataNode.
- Delete one block under data directory.

```
$ hadoop dfs -copyToLocal raidTest/input output.txt
```

You can compare input file *file.txt* with the output file *output.txt*.

3.8 Stop Hadoop and ECPipe

```
$ stop-dfs.sh
$ cd ~ecpipe
$ python scripts/stop.py
```

4 QFS Integration

4.1 Prerequisites

- cmake (v2.8.4 or higher)

```
$ sudo apt-get install cmake
```

- maven (v3.0.3 or higher)

  Download **apache-maven-3.5.0-bin.tar.gz**.

  ```
  $ tar -zxvf apache-maven-3.5.0-bin.tar.gz
  ```

  Set the environment variables `M2_HOME` and `PATH`.

  ```
  export M2_HOME=~/apache-maven-3.5.0
  export PATH=$PATH:$M2_HOME/bin
  ```

- libboost-regex-dev 1.3.4 or higher

  ```
  $ sudo apt-get install libboost-regex-dev
  ```

- libkrb5-dev

  ```
  $ sudo apt-get install libkrb5-dev
  ```

- xfslibs-dev

  ```
  $ sudo apt-get install xfslibs-dev
  ```

- libssl-dev

  ```
  $ sudo apt-get install libssl-dev
  ```

- python-dev

  ```
  $ sudo apt-get install python-dev
  ```

- libfuse-dev

  ```
  $ sudo apt-get install libfuse-dev
  ```

4.2 Install QFS

Download **qfs-1.1.4.tar.gz**.

```
$ tar -zxvf qfs-1.1.4.tar.gz
```

We use `~qfs` to refer to the directory where QFS is installed.

Edit `~ecpipe/qfs-integrate/install.sh` with the proper directory names `~ecpipe` and `~qfs`. Then execute the script.

```
$ ./install.sh
```

4.3 Cluster Setting

We consider erasure coding with $(n, k) = (9, 6)$ in QFS. We configure nine QFS chunkservers and one QFS metaserver.

Node	IP	QFS Node Type	ECPipe Node Type
node1	192.168.0.1	not applicable	coordinator
node2	192.168.0.2	metaserver	helper
node3-node11	192.168.0.3–11	chunkserver	helper

We create the following directories for our test:

- on node2 (metaserver)

```
$ mkdir ~/qfsexe
$ mkdir ~/qfstest
$ mkdir ~/qfstest/conf
$ mkdir ~/qfstest/meta
$ mkdir ~/qfstest/meta/logs
$ mkdir ~/qfstest/meta/checkpoints
```

- on node3-node11 (chunkserver)

```
$ mkdir ~/qfsexe
$ mkdir ~/qfstest
$ mkdir ~/qfstest/conf
$ mkdir ~/qfstest/qfsdata
$ mkdir ~/qfstest/qfslogs
```

We distribute the following executable files to `~/qfsexe`.

```
~/qfs/build/release/bin/metaserver
~/qfs/build/release/bin/chunkserver
~/qfs/build/release/bin/tools/cptoqfs
~/qfs/build/release/bin/tools/cpfromqfs
```

We export the following environment variables.

```
export QFSEXE=~/qfsexe
export PATH=$PATH:$QFSEXE
```

4.4 QFS Configuration

We first configure the metaserver. We provided a sample metaserver configuration file at `~ecpipe/qfs-integrate/conf/MetaServer.example.prp`. You can copy it to `~qfs/qfstest/conf/MetaServer.prp` on the metaserver (node2) and edit it properly. Table 5 explains the details of the configuration file.

Property Name	**Description**
`metaServer.clientPort`	Port number for the metaserver to communicate with clients (e.g., 20000).
`metaServer.chunkServerPort`	Port number for the metaserver to communicate with chunkservers (e.g., 30000).
`metaServer.logDir`	Full path to metaserver logs. (e.g., `~qfs/qfstest/meta/logs`)
`metaServer.cpDir`	Full path to checkpoints. (e.g., `~qfs/qfstest/meta/checkpoints`)
`metaServer.createEmptyFs`	Set to `1` to create an empty file system when QFS is started.
`metaServer.clusterKey`	The clusterKey shared by all nodes.
`metaServer.msgLogWriter.logLevel`	`INFO` or `DEBUG`.
`metaServer.rootDirUser`	The value returned by the command `id -u`.
`metaServer.rootDirGroup`	The value returned by the command `id -g`.
`metaServer.rootDirMode`	0777.

Table 5: Details of the QFS metaserver configuration file.

We next configure each chunkserver. We provided a sample chunkserver configuration file at `~ecpipe/qfs-integrate/conf/ChunkServer.example.prp`. You can copy it to `~qfs/qfstest/conf/ChunkServer.prp` on all chunkservers (node3-node11) and edit it properly. Table 6 explains the details of the configuration file.

4.5 Start QFS

To start the metaserver, you can execute the following on the metaserver:

```
$ metaserver ~qfs/qfstest/conf/MetaServer.prp \
> ~qfs/qfstest/qfslogs/MetaServer.log
```

To start a chunkserver, you can execute the following on each chunkserver:

```
$ chunkserver ~qfs/qfstest/conf/ChunkServer.prp \
> ~qfs/qfstest/qfslogs/ChunkServer.log
```

Property Name	**Description**
`chunkServer.ECPipe.PacketSize`	Slice size in our paper (in bytes).
`chunkServer.ECPipe.PacketCount`	Number of slices in a block.
`chunkServer.metaServer.hostname`	e.g., `192.168.0.2`
`chunkServer.metaServer.port`	e.g., `30000`
`chunkServer.clientPort`	Port number for the chunkserver to communicate with clients (e.g., 22000)
`chunkServer.clusterKey`	The common cluster key.
`chunkServer.chunkDir`	Full path to qfsdata. (e.g., `~qfs/qfstest/qfsdata`)
`chunkServer.pidFile`	Full path to chunkserver.pid (e.g., `~qfs/qfstest/qfslogs/chunkserver.pid`)
`chunkServer.ECPipe.coordinatorIP`	e.g., 192.168.0.1
`chunkServer.ECPipe.localIP`	IP address of the chunkserver itself.
`chunkServer.ECPipe.degradedReadPolicy`	Only `ecpipe` is supported now.
`chunkServer.ECPipe.ECPipePolicy`	Only `basic` is supported now.

Table 6: Details of the QFS chunkserver configuration file.

We can create an input file using `dd` (see our Hadoop-20 test).

```
$ cptoqfs -s 192.168.0.2 -p 20000 -k /path/to/file.txt \
> -d file.txt -S
```

4.6 Start ECPipe

- config.xml

  Table 7 shows the property settings for our QFS test.

- Start ECPipe

```
$ cd ~ecpipe
$ python scripts/start.py
```

Property Name	**Description**
`erasure.code.k`	6 for QFS test.
`erasure.code.n`	9 for QFS test.
`rs.code.config.file`	rsEncMat_6_9
`degraded.read.policy`	Only `ecpipe` is supported now.
`ecpipe.policy`	Only `basic` is supported now.
`file.system.type`	QFS
`block.directory`	QFS block directory.

Table 7: Property settings of config.xml for our QFS test.

4.7 Degraded Read Test

- Ssh to one of the chunkserver nodes.
- Delete one block under data directory.

```
$ cpfromqfs -s 192.168.0.2 -p 20000 -k file.txt -d ./output.txt
```

You can compare the input file with the output file.

4.8 Stop QFS and ECPipe

To stop QFS, first ssh to node2 to stop metaserver.

```
$ killall metaserver
```

Then ssh to node3-node11 to stop chunkserver.

```
$ killall chunkserver
```

To stop ECPipe, ssh to node1:

```
$ cd ~ecpipe
$ python scripts/stop.py
```