# OpenEC v1.0.0 User Guide

ADSLab @ CUHK

Release: Feb 2019

## Contents

**Abstract**

In this user guide, we explain how to install and run OpenEC atop existing distributed storage systems (DSSs). We first explain the preparation steps for running OpenEC (§**??**). We next explain how to integrate OpenEC with HDFS-3 (§**??**), HDFS-RAID (§**??**), and QFS (§**??**). We then explain how to issue basic operations via OpenEC, including writes, reads (both normal and degraded reads), and recovery. Finally, we show how we can add a new erasure code via OpenEC. Please refer to our FAST'19 paper for the design details of OpenEC.

# 1 Preparation

OpenEC has been tested in Ubuntu 14.04. We create a user *openec* and install the packages under its home directory `/home/openec`. You may need the `sudo` access in order to install some of the packages via `apt-get`.

Before installing OpenEC, please first install the following prerequisite libraries.

- cmake v3.1 or higher

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

- g++ v4.8.4

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

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

- redis v3.2.8 or higher

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

```
$ 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 service redis_6379 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 service redis_6379 start
```

- hiredis

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

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

- gf-complete v1.03

  Download and install **gf-complete.tar.gz**. Note that you may need to first install `autoconf` and `libtool`.

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

- ISA-L v2.14.0 or higher

  Download and install **isa-l-2.14.0.tar.gz**. Note that you may need to first install `yasm`, which is required by ISA-L.

```
$ tar -zxvf isa-l-2.14.0.tar.gz
$ cd isa-l-2.14.0
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```

# 2 OpenEC with HDFS-3

## 2.1 Prerequisites

The following packages need to be first installed in order to run HDFS-3.

- maven v3.5.0 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`. You may also need to set `MVN_OPTS` if you are behind a proxy.

  ```
  export M2_HOME=/home/openec/apache-maven-3.5.0
  export PATH=$PATH:$M2_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`.

## 2.2 Install HDFS-3 with OpenEC

Download **hadoop-3.0.0-src.tar.gz** (a copy is available on our project website) and extract the source code to `/home/openec`.

```
$ tar -zxvf hadoop-3.0.0-src.tar.gz
```

We configure the environment variables for HDFS-3. It is recommended to include the following configuration in `~/.bashrc`.

```
export HADOOP_SRC_DIR=/home/openec/hadoop-3.0.0-src
export HADOOP_HOME=$HADOOP_SRC_DIR/hadoop-dist/target/hadoop-3.0.0
export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH
export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar:$HADOOP_CLASSPATH
export CLASSPATH=$JAVA_HOME/lib:$CLASSPATH
export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native:$JAVA_HOME/jre/lib/
amd64/server/:/usr/local/lib:$LD_LIBRARY_PATH
```

Download **openec-v1.0.0.tar.gz** from our project website and extract the source code to `/home/openec`. We can install the patch of OpenEC into HDFS-3 by simply running the script *install.sh*. The script will also compile the modified source code of HDFS-3.

```
$ tar -zxvf openec-v1.0.0.tar.gz
$ cd openec-v1.0.0/hdfs3-integration
$ ./install.sh
```

Please run the following commands to compile OpenEC for HDFS-3.

```
$ cd openec-v1.0.0
$ cmake .  -DFS_TYPE:STRING=HDFS3
$ make
```

## 2.3 Example Architecture

Table **??** shows an example architecture for our HDFS-3 integration. The OpenEC controller runs in the same node as the HDFS-3 NameNode. Each HDFS-3 DataNode is co-located with an OpenEC agent. Please distribute the working directories (`~/hadoop-3.0.0-src` and `~/openec-v1.0.0`) to all the nodes in the testbed.

| IP | HDFS3 | OpenEC |
|---|---|---|
| 192.168.0.1 | NameNode | Controller |
| 192.168.0.2 | DataNode | Agent |
| 192.168.0.3 | DataNode | Agent |
| 192.168.0.4 | DataNode | Agent |
| 192.168.0.5 | DataNode | Agent |

Table 1: Example architecture for HDFS-3 integration.

## 2.4 HDFS-3 Configuration

We provide sample configuration files under `openec-v1.0.0/hdfsraid-integration/conf` for HDFS-3. Here, we show some of the fields related to the integration of OpenEC. You may leave other fields to be the same as our sample configurations. You can copy our sample configuration files to `HADOOP_HOME/etc/hadoop` and configure your HDFS-3 there. Please distribute the configuration files to all the nodes in the testbed.

- hadoop-env.sh:

| Field | Default | Description |
|---|---|---|
| JAVA_HOME | - | Path to java installation.<br>e.g. `/usr/lib/jvm/java-8-oracle` |
| HADOOP_CLASSPATH | $HADOOP_HOME/oeclib/*:<br>$JAVA_HOME/lib* | Path to OpenEC and java libraries. |

- core-site.xml:

| Field | Default | Description |
|---|---|---|
| fs.defaultFS | hdfs://192.168.0.1:9000 | NameNode configuration. |
| hadoop.tmp.dir | /home/openec/hadoop-3.0.0-src/hadoop-dist/<br>target/hadoop-3.0.0 | Base directory for hdfs3 temporary directories. |

- hdfs-site.xml:

| Field | Default | Description |
|---|---|---|
| dfs.replication | 1 | Replication factor of HDFS. |
| dfs.blocksize | 1048576 | The size of a block in bytes. |
| dfs.block.replicator.classname | org.apache.hadoop.hdfs.server.<br>blockmanagement.<br>BlockPlacementPolicyOEC | OpenEC placement integartion. |
| link.oec | true | true: Run HDFS3 with OpenEC.<br>false: Run HDFS3 without OpenEC. |
| oec.controller.addr | 192.168.0.1 | IP address of OpenEC controller. |
| oec.local.addr | - | IP address of a node itself. |
| oec.pktsize | 131072 | The size of a packet in OpenEC. |

- workers:

| |
|---|
| 192.168.0.2 |
| 192.168.0.3 |
| 192.168.0.4 |
| 192.168.0.5 |

To start HDFS-3, we run the following commands in the NameNode.

```
$ hdfs namenode -format
$ start-dfs.sh
```

# 3 OpenEC with HDFS-RAID

## 3.1 Prerequisites

The following packages need first to be installed.

- ant

  Download **apache-ant-1.9.13-bin.tar.gz**.

  ```
  $ tar -zxvf apache-ant-1.9.13.-bin.tar.gz
  ```

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

  ```
  export ANT_HOME=~/apache-ant-1.9.13
  export PATH=$PATH:$ANT_HOME/bin
  ```

- java8

  If you have installed java8, please skip this step.

  ```
  $ 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`.

## 3.2 Install HDFS-RAID with OpenEC

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

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

We configure the environment variables for HDFS-RAID. It is recommended to include the following configuration in `~/.bashrc`.

```
export HADOOP_HOME=/home/openec/hadoop-20
export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH
export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar:$HADOOP_CLASSPATH
export CLASSPATH=$JAVA_HOME/lib:$CLASSPATH
export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native:$JAVA_HOME/jre/lib/
amd64/server/:/usr/local/lib:$LD_LIBRARY_PATH
```

Download **openec-v1.0.0.tar.gz** from our project website and extract the source code to `/home/openec`. We can install the patch of OpenEC into HDFS-RAID by simply running the script *install.sh*. The script will also compile the modified source code of HDFS-RAID.

```
$ tar -zxvf openec-v1.0.0.tar.gz
$ cd openec-v1.0.0/hdfs3-integration
$ ./install.sh
```

We now compile the source code of OpenEC. Please run the following commands.

```
$ cd openec-v1.0.0
$ cmake .  -DFS_TYPE:STRING=HDFSRAID
$ make
```

## 3.3 Example Architecture

Table **??** shows an example architecture for our HDFS-RAID integration. The OpenEC controller runs in the same node as the HDFS-RAID NameNode. Each HDFS-RAID DataNode is co-located with an OpenEC agent. Please distribute the working directories (`~/hadoop-20` and `~/openec-v1.0.0`) to all the nodes in the testbed.

| IP | HDFS-RAID | OpenEC |
|---|---|---|
| 192.168.0.1 | NameNode | Controller |
| 192.168.0.2 | DataNode | Agent |
| 192.168.0.3 | DataNode | Agent |
| 192.168.0.4 | DataNode | Agent |
| 192.168.0.5 | DataNode | Agent |

Table 2: Example architecture for HDFS-RAID integration.

## 3.4 HDFS-RAID Configuration

We provide sample configuration files under `openec-v1.0.0/hdfsraid-integration/conf` for HDFS-RAID. Here, we show some of the fields in detail. You may leave other fields to be the same as our sample configurations. You can copy our sample configuration files to `HADOOP_HOME/conf` and configure your HDFS-RAID there. Please distribute the configuration files to all the nodes in the testbed.

- hadoop-env.sh:

| Field | Default | Description |
|---|---|---|
| JAVA_HOME | - | Path to java installation.<br>e.g. `/usr/lib/jvm/java-8-oracle` |
| HADOOP_CLASSPATH | $HADOOP_HOME/oeclib/*:<br>$JAVA_HOME/lib* | Path to OpenEC and java libraries. |

- core-site.xml:

| Field | Default | Description |
|---|---|---|
| fs.default.name | hdfs://192.168.0.1:9000 | NameNode configuration. |
| hadoop.tmp.dir | /home/openec/hadoop-20/tmp | Base directory for HDFS-RAID temporary directories. |
| topology.script.file.name | - | Path to rackAware.sh |

- hdfs-site.xml:

| Field | Default | Description |
|---|---|---|
| dfs.http.address | 192.168.0.1:50070 | HTTP address of NameNode |
| dfs.replication | 1 | Replication factor of HDFS-RAID. |
| dfs.block.size | 1048576 | The size of a block in bytes. |
| dfs.block.replicator.classname | org.apache.hadoop.hdfs.server.<br>namenode.BlockPlacementPolicyOEC | OpenEC placement class. |
| link.oec | true | Run HDFS-RAID with OpenEC. |
| oec.controller.addr | 192.168.0.1 | IP address of OpenEC controller. |
| oec.local.addr | - | IP address of a node itself. |
| oec.pktsize | 131072 | The size of a packet in OpenEC. |

- masters:

| 192.168.0.1 |
|---|

- slaves:

| 192.168.0.2 |
|---|
| 192.168.0.3 |
| 192.168.0.4 |
| 192.168.0.5 |

To start HDFS-RAID, we run the following commands in the NameNode.

```
$ hadoop namenode -format
$ start-dfs.sh
```

# 4 OpenEC with QFS

## 4.1 Prerequisites

The following packages need to be first installed.

- 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 with OpenEC

Download **qfs-v2.1.1.tar.gz** (a copy is available on our project website).

```
$ tar -zxvf qfs-2.1.1.tar.gz
```

We configure the environment variables for QFS. It is recommended to include the following configuration in `~/.bashrc`.

```
export export QFS_HOME=/home/openec/qfs
export PATH=$QFS_HOME/build/release/bin:$PATH
```

Download **openec-v1.0.0.tar.gz** from our project website and extract the source code to `/home/openec`. We can install the patch of OpenEC into QFS by simply running the script *install.sh*. The script will also compile the modified source code of QFS.

```
$ tar -zxvf openec-v1.0.0.tar.gz
$ cd openec-v1.0.0/qfs-integration
$ ./install.sh
```

We now compile the source code of OpenEC. Please run the following commands.

```
$ cd openec-v1.0.0
$ cmake .  -DFS_TYPE:STRING=QFS
$ make
```

## 4.3 Example Architecture

Table **??** shows an example architecture for our QFS integration. The OpenEC controller runs in the same node as the QFS metaserver. Each QFS chunkserver is co-located with an OpenEC agent. Please distribute the working directories (`~/qfs` and `~/openec-v1.0.0`) to all the nodes in the testbed.

| IP | QFS | OpenEC |
|---|---|---|
| 192.168.0.1 | Metaserver | Controller |
| 192.168.0.2 | Chunkserver | Agent |
| 192.168.0.3 | Chunkserver | Agent |
| 192.168.0.4 | Chunkserver | Agent |
| 192.168.0.5 | Chunkserver | Agent |

Table 3: Example architecture for QFS integration.

## 4.4 QFS Configuration

We also provide sample configuration files for QFS. We show some of the fields in detail and other fields can be the same as in our samples.

- MetaServer.conf:

| Field | Default | Description |
|---|---|---|
| metaServer.clientPort | 20000 | Port number for metaserver to communicate with clients. |
| metaServer.chunkServerPort | 30000 | Port number for metaserver to communicate with chunkserver. |
| metaServer.logDir | - | Directory for the log of metaserver. |
| metaServer.cpDir | - | Directory for the checkpoint of metaserver. |
| openec.useoec | true | true: enable OpenEC integrations.<br>false; disable OpenEC integrations. |
| openec.localip | - | |
| openec.coorip | 192.168.0.1 | IP address of OpenEC controller. |

- ChunkServer.conf:

| Field | Default | Description |
|---|---|---|
| chunkServer.metaServer.hostname | 192.168.0.1 | IP address of metaserver. |
| chunkServer.metaServer.port | 30000 | Port number for metaserver to communicate with chunkserver. |
| chunkServer.clientPort | 22000 | Port number for chunkserver to communicate with clients. |
| chunkServer.chunkDir | - | Directory for chunkserver to store chunks. |
| chunkServer.pidFile | - | File to store pid of chunkserver |
| openec.useoec | true | true: enable OpenEC integrations.<br>false; disable OpenEC integrations. |
| openec.localip | - | |
| openec.coorip | 192.168.0.1 | IP address of OpenEC controller. |

To start the metaserver, please run the following command:

```
$ metaserver MetaServer.conf
```

To start each chunkserver, please run the following command:

```
$ chunkserver ChunkServer.conf
```

# 5 OpenEC Configuration

We provide sample configuration files for OpenEC under `openec-v1.0.0/conf`. Table **??** explains the default configuration in our sample. Table **??** and Table **??** show the configuration of an erasure code and an offline encoding pool, respectively.

| Field | Default | Description |
|---|---|---|
| controller.address | 192.168.0.1 | IP address of controller. |
| agents.address | /default/192.168.0.2<br>/default/192.168.0.3<br>/default/192.168.0.4<br>/default/192.168.0.5 | A list of IP addresses of all agents, in the form of *zone/IP*, where *zone* denotes the zone (e.g. rack or datacenter). |
| local.address | - | IP address of a node itself. |
| packet.size | 131072 | The size of a packet. |
| dss.type | - | Type of DSS. Please choose from *HDFS3*, *HDFSRAID* and *QFS*. |
| dss.parameter | - | IP and port of DSS for client access. e.g. *192.168.0.1, 9000* for HDFS3. |
| ec.policy | | Table **??** |
| offline.pool | | Table **??** |

Table 4: sysSetting.xml for OpenEC

To start OpenEC, we run the following command in controller:

| Field | Default | Description |
|---|---|---|
| ecid | rs_4_3 | Unique id for an erasure code. |
| class | RSCONV | Class name of erasure code implementation. |
| n | 4 | Parameter N for the erasure code. |
| k | 3 | Parameter K for the erasure code. |
| w | 1 | Parameter W for the erasure code. |
| opt | -1 | Optimization level for OpenEC. Four levels of optimization is provided by OpenEC, including -1, 0, 1, 2. -1: no optimization are enabled. 0: BindX is enabled. 1: BindX and BindY are enabled. 2: Hierarchical awareness is enabled. |

Table 5: ec.policy configuration

| Field | Default | Description |
|---|---|---|
| poolid | rs_4_3_pool | Unique id for an offline encoding pool. |
| ecid | rs_4_3 | Erasure code that is applied for the pool. |
| base | 1 | Block size (in MiB) for the pool, which is no larger than the block size in HDFS3. |

Table 6: offline.pool configuration

```
$ cd openec-v1.0.0
$ python script/start.py
```

# 6 Basic Operations

We explain how to issue writes, reads (normal and degraded reads), and recovery via OpenEC.

## 6.1 Write

OpenEC supports two modes to write a file into a DSS: (i) writing a file with online encoding enabled on the writing path; and (ii) writing a file into an offline encoding pool, in which a coding group is organized and encoded offline.

We run OECClient to issue a write request. We show the usage and a command-line example to write a file (called *input*) of size 3 MiB and store it as */testfile1* with online encoding enabled. The erasure code for online encoding is *rs_4_3* (i.e., RS codes with $n = 4$ and $k = 3$), which is configured in `sysSetting.xml`. Please note that this command should run in a node that holds an Agent.

Usage:

```
./OECClient write [inputfile] [saveas] [ecid] online [sizeinMB]
```

Example:

```
$ ./OECClient write input /testfile1 rs_4_3 online 3
```

We now show how to apply offline encoding. We first write the file into our offline encoding pool. The command-line example in the following shows that we write the file (*input*) and store it as */testfile2*. The offline encoding pool is *rs_4_3_pool*, which is configured in `sysSetting.xml`. Please note that this command should also run in a node that holds an Agent.

Usage:

```
./OECClient write [inputfile] [saveas] [poolid] offline [sizeinMB]
```

Example:

```
$ ./OECClient write input /testfile2 rs_4_3_pool offline 3
```

We then run the following command to instruct OpenEC to start offline encoding.

```
$ ./OECClient startEncode
```

When the offline encoding for a coding group finishes, we can check the log of the controller (*coor_output*). The following line means that the offline encoding for a coding group finishes, where *xxxxxx* denotes the name of the coding group. Note that the name of a coding group is assigned by OpenEC.

```
offlineEnc for xxxxxx finishes
```

## 6.2 Read

We run OECClient to read a file, either a normal read or a degraded read. The difference between a normal read and a degraded read is that for a degraded read, some physical blocks in the DSS are unavailable (e.g., deleted) before the read request is issued. Please note that this command should also run in a node that holds an Agent.

Usage:

```
./OECClient read [filename] [saveas]
```

Example:

```
$ ./OECClient read /testfile1 output1
$ ./OECClient read /testfile2 output2
```

### 6.3 Recovery

For recovery, we can delete some physical blocks in the DSS and then run the following command to instruct OpenEC to repair the lost blocks.

```
$ ./OECClient startRepair
```

We can see the following information in the log of controller, which denotes that OpenEC finishes repairing a block. Note that *xxxxxx* is the corresponding block name in OpenEC.

```
repair for xxxxxx finishes
```

## 7 EC Design in OpenEC

We introduce how to design a new erasure code in OpenEC. The following shows the base class of an erasure code implementation in OpenEC. To add a new erasure code, we need to extend this base class and provide the implementations for *Encode*, *Decode* and *Place* methods. We provide several erasure code implementations under `openec-v1.0.0/src/ec`. Please refer to our sample implementations there.

```
class ECBase {
public:
int _n, _k, _w;
int _opt;

ECBase();
ECBase(int n, int k, int w, int opt, vector<string> param);

virtual ECDAG* Encode() = 0;
virtual ECDAG* Decode(vector<int> from, vector<int> to) = 0;
virtual void Place(vector<vector<int>>& group) = 0;
};
```