This source code is a simple implementation to evaluate the delta-based update in erasure coded data centers. Please refer to the paper: *Cross-Rack-Aware Updates in Erasure-Coded Data Centers* (in Proceedings of 47th International Conference on Parallel Processing (**ICPP'18**)) for more technical details.

In what follows, we will describe how to install and run CAU code on Ubuntu. Before running the code, please make the following preparations first.

Preparations

1: ensure that the necessary libraries and compile tools are installed, including gcc and make.

```
$ sudo apt-get install gcc make
```

2: change the default configurations in "config.h", including the erasure code you wish to use, and the data center architecture (e.g., the number of racks, and the number of nodes in each rack). Notice that we currently assume all the racks are composed of the same number of nodes. The next figure shows an example of our configurations, in which we use RS(4,2) code, and the cluster is composed of two racks with two nodes per rack.

```
/* if the server to act as gateway switch is opened */
#define GTWY_OPEN            1

// ======================== Fill the erasure coding parameters ====================
/* erasure coding settings */
#define data_chunks          2
#define num_chunks_in_stripe 4
#define chunk_size           1024*1024 // suppose the chunk size is 1MB

// ======================== Fill the architecture parameters ====================

/* configurations of the data center */
#define total_nodes_num      4 //we set the number of chunks as the number of nodes in the local-cluster evaluation
#define max_chunks_per_rack  2
#define rack_num             2
#define node_num_per_rack    total_nodes_num/rack_num //we currently assume that each rack is composed of a constant number of nodes

// ============================ END ============================================
```

3: generate a big file named "data_file" which will be used for simulating disk reads and writes. In our test, we generally set its size as 60GB.

4: In the code, we need to read the ip address from network interface . The network interface name in our testbed is "enp0s31f6" (see the global variable "NIC" in common.c). If your machine has a different network interface name, please replace "enp0s31f6" with it. You can use the "ifconfig" command to see the network interface name and the inner ip address of a node. The following example shows that the default network interface name of the node is "enp0s31f6" and its inner ip address is 192.168.0.51.

```
$ ifconfig
enp0s31f6 Link encap:Ethernet  HWaddr 4c:cc:6a:e3:de:3b
          inet addr:192.168.0.51  Bcast:192.168.0.255  Mask:255.255.255.0
```

For example, if the default network interface name of your node is "eth0", then you can replace "enp0s31f6" with "eth0" by assigning "eth0" to the global variable "NIC" in common.c.

```
// =============== Fill the ip addresses of the nodes in the evaluation =============

/* it records the public ip address in socket communications */
char* node_ip_set[total_nodes_num]={"192.168.10.53", "192.168.10.54", "192.168.10.55", "192.168.10.56"};

/* it records the inner ip address read from NIC */
char* inner_ip_set[total_nodes_num]={"192.168.0.53", "192.168.0.54", "192.168.0.55", "192.168.0.56"};

/* public ip of the metadata server */
char* mt_svr_ip="192.168.10.52";

/* public ip of the client */
char* client_ip="192.168.10.51";

/* default NIC */
char* NIC="enp0s31f6";

/* public ip of the gateway server */
char* gateway_ip="192.168.10.58";
/* inner ip of the gateway server */
char* gateway_local_ip="192.168.0.58";

// =============== Fill the number of nodes in each rack in the evaluation ===========

/* number of nodes in each rack */
int   nodes_in_racks[rack_num]={node_num_per_rack, node_num_per_rack};

/* rack names */
char* region_name[rack_num]={"Rack 0", "Rack 1"};

// ================================ END ============================================
```

5: two kinds of ip addresses are needed in the evaluation: inner ip and public ip. "Inner-ip" denotes the ip address read from the default network interface, while "public ip" denotes the ip address used in socket communications.

In this step, you should fill the two kinds of ip addresses you will use in the file common.c.

```
// =============== Fill the ip addresses of the nodes in the evaluation =============

/* it records the public ip address in socket communications */
char* node_ip_set[total_nodes_num]={"192.168.10.53", "192.168.10.54", "192.168.10.55", "192.168.10.56"};

/* it records the inner ip address read from NIC */
char* inner_ip_set[total_nodes_num]={"192.168.0.53", "192.168.0.54", "192.168.0.55", "192.168.0.56"};

/* public ip of the metadata server */
char* mt_svr_ip="192.168.10.52";

/* public ip of the client */
char* client_ip="192.168.10.51";

/* default NIC */
char* NIC="enp0s31f6";

/* public ip of the gateway server */
char* gateway_ip="192.168.10.58";
/* inner ip of the gateway server */
char* gateway_local_ip="192.168.0.58";

// =============== Fill the number of nodes in each rack in the evaluation ===========

/* number of nodes in each rack */
int   nodes_in_racks[rack_num]={node_num_per_rack, node_num_per_rack};

/* rack names */
char* region_name[rack_num]={"Rack 0", "Rack 1"};

// ================================ END ============================================
```

6: fill in the **public ip addresses** of the metadata server and the client.

```
// =============== Fill the ip addresses of the nodes in the evaluation =============

/* it records the public ip address in socket communications */
char* node_ip_set[total_nodes_num]={"192.168.10.53", "192.168.10.54", "192.168.10.55", "192.168.10.56"};

/* it records the inner ip address read from NIC */
char* inner_ip_set[total_nodes_num]={"192.168.0.53", "192.168.0.54", "192.168.0.55", "192.168.0.56"};

/* public ip of the metadata server */
char* mt_svr_ip="192.168.10.52";

/* public ip of the client */
char* client_ip="192.168.10.51";

/* default NIC */
char* NIC="enp0s31f6";

/* public ip of the gateway server */
char* gateway_ip="192.168.10.58";
/* inner ip of the gateway server */
char* gateway_local_ip="192.168.0.58";

// =============== Fill the number of nodes in each rack in the evaluation ===========

/* number of nodes in each rack */
int   nodes_in_racks[rack_num]={node_num_per_rack, node_num_per_rack};

/* rack names */
char* region_name[rack_num]={"Rack 0", "Rack 1"};

// ================================ END ============================================
```

7: fill the architecture information including the number of nodes per rack and the rack names.

```
// =============== Fill the ip addresses of the nodes in the evaluation =============

/* it records the public ip address in socket communications */
char* node_ip_set[total_nodes_num]={"192.168.10.53", "192.168.10.54", "192.168.10.55", "192.168.10.56"};

/* it records the inner ip address read from NIC */
char* inner_ip_set[total_nodes_num]={"192.168.0.53", "192.168.0.54", "192.168.0.55", "192.168.0.56"};

/* public ip of the metadata server */
char* mt_svr_ip="192.168.10.52";

/* public ip of the client */
char* client_ip="192.168.10.51";

/* default NIC */
char* NIC="enp0s31f6";

/* public ip of the gateway server */
char* gateway_ip="192.168.10.58";
/* inner ip of the gateway server */
char* gateway_local_ip="192.168.0.58";

// =============== Fill the number of nodes in each rack in the evaluation ===========

/* number of nodes in each rack */
int   nodes_in_racks[rack_num]={node_num_per_rack, node_num_per_rack};

/* rack names */
char* region_name[rack_num]={"Rack 0", "Rack 1"};

// ================================ END ============================================
```

8: In our evaluation, we use a gateway server to mimic cross-rack data transfers in a local cluster (see our paper for more details). If you wish to do this, take the following two steps:

- set the GTWY_OPEN as 1 in config.h

- set the gateway_ip (the public ip for socket communication) and the gateway_local_ip (the inner ip read from network interface) in common.c

```
// =============== Fill the ip addresses of the nodes in the evaluation =============

/* it records the public ip address in socket communications */
char* node_ip_set[total_nodes_num]={"192.168.10.53", "192.168.10.54", "192.168.10.55", "192.168.10.56"};

/* it records the inner ip address read from NIC */
char* inner_ip_set[total_nodes_num]={"192.168.0.53", "192.168.0.54", "192.168.0.55", "192.168.0.56"};

/* public ip of the metadata server */
char* mt_svr_ip="192.168.10.52";

/* public ip of the client */
char* client_ip="192.168.10.51";

/* default NIC */
char* NIC="enp0s31f6";

/* public ip of the gateway server */
char* gateway_ip="192.168.10.58";
/* inner ip of the gateway server */
char* gateway_local_ip="192.168.0.58";

// =============== Fill the number of nodes in each rack in the evaluation ===========

/* number of nodes in each rack */
int   nodes_in_racks[rack_num]={node_num_per_rack, node_num_per_rack};

/* rack names */
char* region_name[rack_num]={"Rack 0", "Rack 1"};

// ================================ END ============================================
```

9: If you wish to deploy the code onto Amazon EC2, please do the following two things.

- carefully specify the "security group" by only allowing the communications among the VMs used in the test. We have ever encountered unexpected connections (may be from other VMs), which will definitely affect the running status of evaluations.
- each VM in Amazon EC2 has two ip addresses, the public ip and the inner ip. You have to fill them in common.c.

An example of running CAU code:

After filling the configuration information, we will show how to run CAU code in next steps. The running of the baseline delta-based update approach and the PARIX is similar.

- extract the files from cau-1.0.0.tar

```
$ tar zxvf cau-1.0.0.tar
$ cd cau-1.0.0/
$ export CAU_HOME=$(pwd)
```

- generate the needed object files of Jerasure

```
$ cd ${CAU_HOME}/Jerasure
$ make
```

- generate the executable files in CAU

```
$ cd ${CAU_HOME}
$ make
```

- run "gen_chunk_distribn" on the metadata server (MDS), which will generate the mapping information between the logical chunks and the associated storage nodes. The mapping information will be recorded in a file named "chunk_map" in the MDS. The MDS will read it for chunk addressing. Notice that the mapping information is generated based on the selected erasure coding and the data center architecture specified in "config.h".

```
$ cd  ${CAU_HOME}
$ ./gen_chunk_distribn
```

- copy the executable files with the suffix of "_mds" and the "chunk_map" file to the MDS.
- copy the executable files with the suffix of "_server" to storage nodes (including the gateway server if enabled).
- run the executable files with the suffix of "_mds" (e.g., cau_mds) on MDS

```
$ cd ${CAU_HOME}
$ ./cau_mds
```

- run the executable files with the suffix of "_server" on storage nodes (including data nodes, parity nodes, and the gateway server if enabled).

```
$ cd ${CAU_HOME}
$ ./cau_server
```

- run the executable file with the suffix "_client" on the client with the trace file to evaluate. Some example traces are included in "example-traces"

```
$ cd ${CAU_HOME}
$ ./cau_client example-traces/wdev_1.csv
```

If you have any question, please feel free to contact me (zhirong.shen2601@gmail.com).