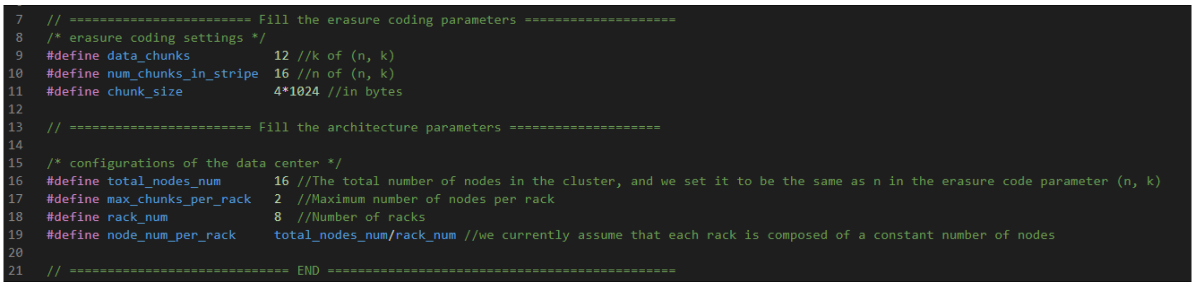
This is a simple implementation to evaluate Rack-Coordinated Updates (RackCU) in erasure-coded data centers. The following shows how to run RackCU in Ubuntu.

Preparations

1: Download gcc and make

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

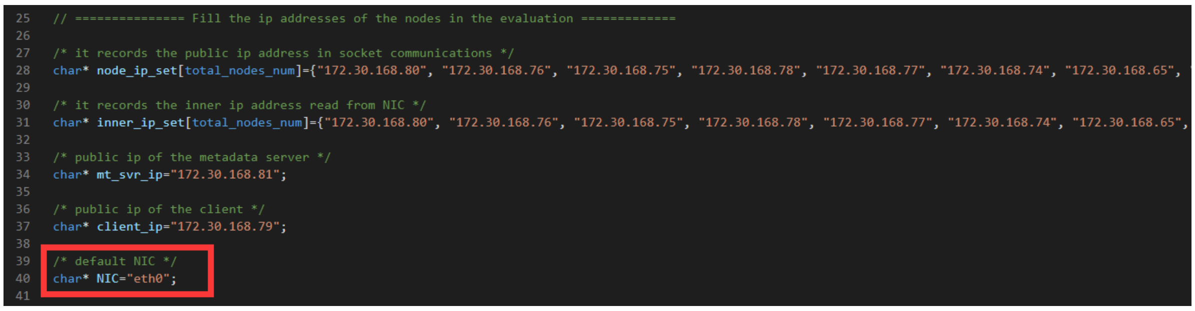
2: Change the default configuration in "config.h". The figure below is an example of our configuration. The meaning of each parameter has been given in the comments of the code. You need to modify them according to your own experimental environment.



```
// ====================== Fill the erasure coding parameters ===================
/* erasure coding settings */
#define data_chunks           12 //k of (n, k)
#define num_chunks_in_stripe  16 //n of (n, k)
#define chunk_size            4*1024 //in bytes

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

/* configurations of the data center */
#define total_nodes_num       16 //The total number of nodes in the cluster, and we set it to be the same as n in the erasure code parameter (n, k)
#define max_chunks_per_rack   2  //Maximum number of nodes per rack
#define rack_num              8  //Number of racks
#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 ==============================================
```

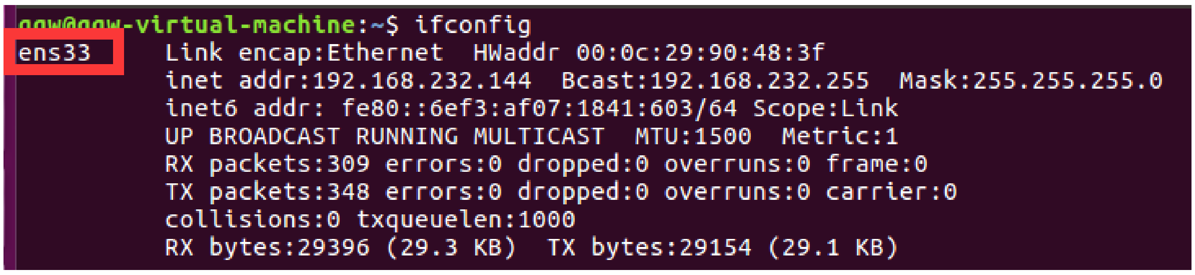
3: Generate a data file named "data_file" and place it in each storage node. In our experiment, we placed a 4GB "data_file" on each storage node.

4: Change network interface name "NIC" in "common.c" based on your network interface.



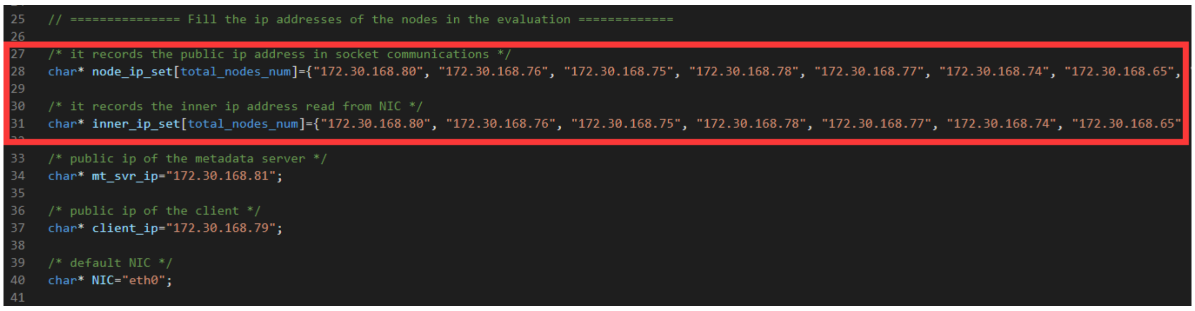
```
// ============== Fill the ip addresses of the nodes in the evaluation ============

/* it records the public ip address in socket communications */
char* node_ip_set[total_nodes_num]={"172.30.168.80", "172.30.168.76", "172.30.168.75", "172.30.168.78", "172.30.168.77", "172.30.168.74", "172.30.168.65",

/* it records the inner ip address read from NIC */
char* inner_ip_set[total_nodes_num]={"172.30.168.80", "172.30.168.76", "172.30.168.75", "172.30.168.78", "172.30.168.77", "172.30.168.74", "172.30.168.65",

/* public ip of the metadata server */
char* mt_svr_ip="172.30.168.81";

/* public ip of the client */
char* client_ip="172.30.168.79";

/* default NIC */
char* NIC="eth0";
```

You can use the "ifconfig" command to see the your network interface name. For example, in the figure below, we can see that the network interface name is "ens33" through the "ifconfig" command, so we need to change the NIC in "commom.c" to "ens33".



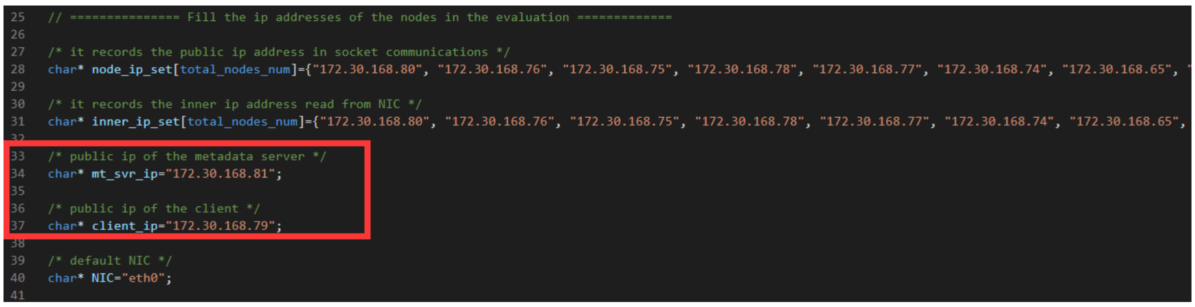
```
gqw@gqw-virtual-machine:~$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:90:48:3f
          inet addr:192.168.232.144  Bcast:192.168.232.255  Mask:255.255.255.0
          inet6 addr: fe80::6ef3:af07:1841:603/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:309 errors:0 dropped:0 overruns:0 frame:0
          TX packets:348 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:29396 (29.3 KB)  TX bytes:29154 (29.1 KB)
```

5: Change the IP address of the storage node. Our experimental environment does not use public IP, set the public IP("node_ip_set[]") and internal IP("inner_ip_set[]") as the same.



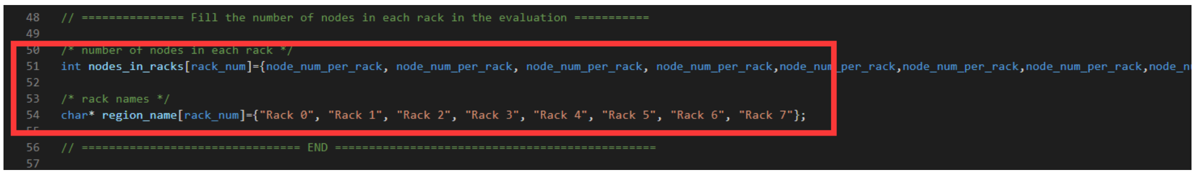
```c
// ============== Fill the ip addresses of the nodes in the evaluation ============

/* it records the public ip address in socket communications */
char* node_ip_set[total_nodes_num]={"172.30.168.80", "172.30.168.76", "172.30.168.75", "172.30.168.78", "172.30.168.77", "172.30.168.74", "172.30.168.65",

/* it records the inner ip address read from NIC */
char* inner_ip_set[total_nodes_num]={"172.30.168.80", "172.30.168.76", "172.30.168.75", "172.30.168.78", "172.30.168.77", "172.30.168.74", "172.30.168.65"

/* public ip of the metadata server */
char* mt_svr_ip="172.30.168.81";

/* public ip of the client */
char* client_ip="172.30.168.79";

/* default NIC */
char* NIC="eth0";
```

6: Change the IP address of metadata server and client according to your environment.



```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]={"172.30.168.80", "172.30.168.76", "172.30.168.75", "172.30.168.78", "172.30.168.77", "172.30.168.74", "172.30.168.65",

/* it records the inner ip address read from NIC */
char* inner_ip_set[total_nodes_num]={"172.30.168.80", "172.30.168.76", "172.30.168.75", "172.30.168.78", "172.30.168.77", "172.30.168.74", "172.30.168.65",

/* public ip of the metadata server */
char* mt_svr_ip="172.30.168.81";

/* public ip of the client */
char* client_ip="172.30.168.79";

/* default NIC */
char* NIC="eth0";
```

7: Fill the architecture information including the number of nodes per rack and the rack names in "common.c".



```c
// ============== 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, node_num_per_rack, node_num_per_rack,node_num_per_rack,node_num_per_rack,node_num_per_rack,node_n

/* rack names */
char* region_name[rack_num]={"Rack 0", "Rack 1", "Rack 2", "Rack 3", "Rack 4", "Rack 5", "Rack 6", "Rack 7"};

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

An example of running RackCU:

1: Add Rackcu source code to your experimental environment.

2: Generate the needed object files of Jerasure

```
$ cd RackCU/Jerasure
$ make
```

3: generate the executable files in RackCU

```
$ cd RackCU/
$ make
```

4: Run "gen_chunk_distribn" on the metadata server (MDS). It will generate the mapping information between the logical chunks and the associated storage nodes. You can see the mapping information in a file named "chunk_map" in MDS.

```
$ cd RackCU/
$ ./gen_chunk_distribn
```

5: Run the executable files with the suffix of "_mds" (e.g., rcu_mds) on MDS

```
$ cd RackCU/
$ ./rcu_mds
```

6: Run the executable files with the suffix of "_server" (e.g., rcu_server) on each storage node (including data nodes and parity nodes).

```
$ cd RackCU/
$ ./rcu_server
```

7: Run the executable file with the suffix "_client" (e.g., rcu_client) on the client with the trace file to evaluate. We have some example traces in "example-traces".

```
$ cd RackCU/
$ ./rcu_client example-traces/wdev_1.csv
```

If you have any question, please feel free to contact me (23020201153743@stu.xmu.edu.cn).