Magistrello Camilla (4512554)

Assignment 2: Peer-to-Peer backup

In this assignment I simulate a peer-to-peer backup application using erasure coding.

To test and observe the obtained results, I created a class called FinalBlock that takes the nodes as input once the simulation is complete. Inside it, I implemented a function that checks the saved blocks for each individual node.

Thanks to this function, it is possible to observe, for each node, whether the data has been saved and where.

```
def finalBlock(self, nodes: List['Node']):
    self.nodes = nodes
    for node in self.nodes:
        if(len(node.local_blocks) > 0):
            print("NODE: ", node.name, " local_blocks " , node.local_blocks)
            self.block_true = []
            self.block_false_saved = []
            i = 0
            for block in node.local_blocks:
                if node.local_blocks[i] == True:
                    self.block_true.append(block)
                if node.backed_up_blocks[i] is not None and not node.local_blocks[i]:
                    self.block_false_saved.append(block)
                i += 1
            for peer, block_id in node.remote_blocks_held.items():
                print("I am saving, for node ", peer, ", block ", block_id)
            blocchiFinaliRecuperati = len(self.block_true) + len(self.block_false_saved)
            print("In node ", node.name, " i managed to recover ", blocchiFinaliRecuperati, " blocks.")
```

To understand which blocks I had not lost during the simulation, I created 2 lists:

- **block_true** where, for each node, I saved the blocks that were locally stored for that node.
- **block_false_saved** where I saved the blocks that are not local but have been saved on another node, and therefore can be recovered.

Testing the simulation with the configuration file p2p.cfg, I obtained these results:

- **NODE: peer-0** *local_blocks* [False, False, False, False, False, True, True, False, False, False]

  I am saving, for node peer-2, block 3.

  I am saving, for node peer-1, block 4.

  I am saving, for node peer-9, block 3.

  I am saving, for node peer-7, block 0.

  I am saving, for node peer-6, block 6.

  I am saving, for node peer-5, block 7.

  I am saving, for node peer-3, block 5.

I am saving, for node peer-8, block 6.

In node peer-0 I managed to recover 2 blocks.

- **NODE: peer-1** *local_blocks* [True, True, True, True, True, True, True, True, True, True]

  I am saving, for node peer-2, block 2.

  I am saving, for node peer-9, block 8.

  I am saving, for node peer-5, block 5.

  I am saving, for node peer-4, block 4.

  I am saving, for node peer-6, block 1.

  I am saving, for node peer-3, block 7.

  In node peer-1 I managed to recover 10 blocks.

- **NODE: peer-2** *local_blocks* [True, True, True, True, True, True, True, True, True, True]

  I am saving, for node peer-9, block 4.

  I am saving, for node peer-1, block 1.

  In node peer-2 I managed to recover 10 blocks.

- **NODE: peer-3** *local_blocks* [False, False, True, True, True, True, False, True, False, False]

  I am saving, for node peer-1, block 6.

  I am saving, for node peer-9, block 2.

  I am saving, for node peer-2, block 7.

  I am saving, for node peer-5, block 6.

  I am saving, for node peer-0, block 6.

  I am saving, for node peer-7, block 1.

  I am saving, for node peer-6, block 5.

  I am saving, for node peer-4, block 5.

  I am saving, for node peer-8, block 2.

  In node peer-3 I managed to recover 5 blocks.

- **NODE: peer-4** *local_blocks* [False, False, True, True, True, True, False, False, True, False]

  I am saving, for node peer-2, block 5.

  I am saving, for node peer-1, block 8.

  I am saving, for node peer-9, block 6.

  I am saving, for node peer-5, block 3.

  I am saving, for node peer-8, block 5.

  I am saving, for node peer-3, block 3.

  I am saving, for node peer-7, block 7.

  In node peer-4 I managed to recover 5 blocks.

- **NODE: peer-5** local_blocks [False, True, False, True, True, True, True, True, False, False]

    I am saving, for node peer-1, block 2.

    I am saving, for node peer-2, block 0.

    I am saving, for node peer-9, block 1.

    I am saving, for node peer-4, block 8.

    I am saving, for node peer-8, block 8.

    In node peer-5 I managed to recover 6 blocks.

- **NODE: peer-6** local_blocks [False, True, False, False, False, True, True, True, False, False]

    I am saving, for node peer-2, block 4.

    I am saving, for node peer-1, block 0.

    I am saving, for node peer-9, block 0.

    I am saving, for node peer-3, block 2.

    I am saving, for node peer-5, block 1.

    I am saving, for node peer-4, block 3.

    In node peer-6 I managed to recover 4 blocks.

- **NODE: peer-7** local_blocks [True, True, False, False, False, False, False, True, False, False]

    I am saving, for node peer-2, block 8.

    I am saving, for node peer-9, block 7.

    I am saving, for node peer-1, block 7.

    I am saving, for node peer-4, block 2.

    I am saving, for node peer-5, block 4.

    I am saving, for node peer-0, block 5.

    In node peer-7 I managed to recover 3 blocks.

- **NODE: peer-8** local_blocks [False, False, True, False, False, True, True, False, True, False]

    I am saving, for node peer-2, block 6.

    I am saving, for node peer-9, block 5.

    I am saving, for node peer-1, block 5.

    I am saving, for node peer-3, block 4.

    I am saving, for node peer-6, block 7.

    In node peer-8 I managed to recover 4 blocks.

- **NODE: peer-9** local_blocks [False, True, True, True, True, False, True, True, True, False].

    I am saving, for node peer-2, block 1.

    I am saving, for node peer-1, block 3.

    In node peer-9 I managed to recover 9 blocks.

From these results, it can be observed how nodes peer-1, peer-2, and peer-9 managed, at the end of the simulation, to retrieve enough blocks to be able to "reconstruct" the initial data thanks to erasure coding.

It can be observed that peer-9 has 7 blocks locally and 2 saved blocks, one on peer-6 and the other on peer-8, thus managing to recover enough blocks to reconstruct the entire initial data.

Regarding the simulation with the configuration file client_server.cfg, the results obtained are, almost always, the complete recovery by the client of its data since it has 10 servers available to save its blocks.

Assignment extensions:

Since nodes in a peer-to-peer (p2p) system are distributed, they are more susceptible to hacker attacks and behaving maliciously compared to a centralized system.

Let's assume that some nodes in our system will be used maliciously. These "malicious" nodes would disguise themselves as other nodes during the backup, saving corrupted and potentially dangerous data.

Two parameters have been added to the Node class:

- NodeBad: containing the list of bad nodes.
- FakeName: the name that the bad node uses to pretend to be another.

In the “client_server.cfg” configuration file, the “nodeBad” field is declared for servers as “nodeBad = /”, since they cannot impersonate other nodes as they do not perform backups. In this test case, the total number of clients has also been increased to test the simulation with the presence of at least one bad client and one good client.

The “fakeName” parameter is always declared as “fakeName = /” in the configuration files and set to None at the time of creating individual nodes.

When the nodes are created, a boolean value is set to True for them only if their original name appears in the “nodeBad” list in the configuration file.

Moreover, a value called "OriginalName" is generated to preserve the actual name of the node.

```
self.fakeName = None
self.originalName = self.name
self.listOfBadPeer = self.nodeBad.split(",")
if self.listOfBadPeer.count(self.originalName):
    self.isABadPeer = True
else:
    self.isABadPeer = False
```

The backup can be performed if:

- Both nodes are good.
- Both nodes are malicious.
- The bad node wants to perform a backup on a good node, pretending to be another node, in order to save false data.

In the schedule_next_upload function, the backup has been modified to:

```
# first find if we have a backup that a remote node needs
for peer, block_id in self.remote_blocks_held.items():
    if peer.online and peer.current_download is None and not peer.local_blocks[block_id]:
        sim.schedule_transfer(self, peer, block_id, restore=True)
        return  # we have found our upload, we stop
block_id = self.find_block_to_back_up()
if block_id is None:
    return
remote_owners = set(node for node in self.backed_up_blocks if node is not None)
for peer in sim.nodes:
    if self.isABadPeer: # self is bad
        if peer.isABadPeer: # peer is bad
            if (peer is not self and peer.online and peer not in remote_owners and peer.current_download is None
            and peer.free_space >= self.block_size):
                sim.schedule_transfer(self, peer, block_id, restore=False)
                return
        elif not peer.isABadPeer: # peer is good
            fake = random.choice(sim.nodes)
            if fake is not self and fake is not peer and not fake.isABadPeer and fake.data_size > 0:
                self.fakeName = fake.name
                if (peer.name is not self.fakeName and peer.online and peer not in remote_owners and peer.current_download is None
                and peer.free_space >= self.block_size):
                    sim.log_info(f"{self.originalName} is a bad node and fake its identity as {fake.name} to back up block {block_id} on {peer.name}")
                    sim.log_info(f"{fake.name} will end up with backups of data that are not his own or are dirty.")
                    sim.schedule_transfer(self, peer, block_id, restore=False)
                    return

    else: # self is good
        if not peer.isABadPeer: # peer is bad
            # downloading anything currently, schedule the backup of block_id from self to peer
            if (peer is not self and peer.online and peer not in remote_owners and peer.current_download is None
                    and peer.free_space >= self.block_size):
                sim.schedule_transfer(self, peer, block_id, restore=False)
                return
```

When a bad node wants to perform a backup while pretending to be another, the value of “fakeName” is set in such a way that in the “update_block_state” function, it is possible to save the fake node name in the “remote_blocks_held” instead of the real one.

```
class BlockBackupComplete(TransferComplete):
    def update_block_state(self):
        owner, peer = self.uploader, self.downloader
        peer.free_space -= owner.block_size
        assert peer.free_space >= 0
        owner.backed_up_blocks[self.block_id] = peer
        if owner.fakeName is not None:
            owner.name = owner.fakeName
        peer.remote_blocks_held[owner] = self.block_id
```

The same approach has been applied to the “schedule_next_download” function, where the value of “fakeName” is utilized to manage the scheduling of the next download, allowing the bad node to simulate the behavior of another node during this process.

```
for block_id, (held_locally, peer) in enumerate(zip(self.local_blocks, self.backed_up_blocks)):
    if not held_locally and peer is not None and peer.online and peer.current_upload is None:
        sim.schedule_transfer(peer, self, block_id, restore=True)
        return  # we are done in this case
for peer in sim.nodes:
    if peer.isABadPeer: # peer is bad
        if self.isABadPeer: # self is bad
            if (peer is not self and peer.online and peer.current_upload is None and peer not in self.remote_blocks_held
            and self.free_space >= self.block_size):
                block_id = peer.find_block_to_back_up()
                if block_id is not None:
                    sim.schedule_transfer(peer, self, block_id, restore=False)
                    return
        elif not self.isABadPeer: # self is good
            fake = random.choice(sim.nodes)
            if fake is not peer and not fake.isABadPeer and fake.data_size > 0:
                peer.fakeName = fake.name
                if (peer.fakeName is not self.name and fake is not peer and peer.online and peer.current_upload is None and peer.fakeName not in self.remote_blocks_held
                and self.free_space >= peer.block_size):
                    block_id = peer.find_block_to_back_up()
                    if block_id is not None:
                        sim.log_info(f"{peer.originalName} is a bad node and fake its identity as {peer.fakeName} to back up block {block_id} on {self.name}")
                        sim.log_info(f"{fake.name} will end up with backups of data that are not his own or are dirty.")
                        sim.schedule_transfer(peer, self, block_id, restore=False)
                        return
    else:
        if not self.isABadPeer: # self is good
            # if the peer is not self, is online, is not among the remote owners, has enough space and is not
            # downloading anything currently, schedule the backup of block_id from self to peer
            if (peer is not self and peer.online and peer.current_upload is None and peer not in self.remote_blocks_held
            and self.free_space >= self.block_size):
                block_id = peer.find_block_to_back_up()
                if block_id is not None:
                    sim.schedule_transfer(peer, self, block_id, restore=False)
                    return
```

To test and observe the obtained results, the FinalBlock class has been slightly modified, allowing for the observation of both the original names and the fake names of the nodes:

```
class FinalBlock(Simulation):

    def __init__(self, nodes: List['Node']):
        super().__init__()  # call the __init__ method of parent class
        self.nodes = nodes
        self.finalBlock(self.nodes)

    def finalBlock(self, nodes: List['Node']):
        self.nodes = nodes
        for node in self.nodes:
            print("NODE:", node.originalName, "local_blocks" , node.local_blocks)
            self.block_true = []
            self.block_false_saved = []
            i = 0
            for block in node.local_blocks:
                if node.local_blocks[i] == True:
                    self.block_true.append(block)
                if node.backed_up_blocks[i] is not None and not node.local_blocks[i]:
                    self.block_false_saved.append(block)
                i += 1

            for peer, block_id in node.remote_blocks_held.items():
                if node.isABadPeer:
                    print("I am saving, for node", peer.originalName,", block", block_id,".")
                else:
                    print("I am saving, for node", peer.name,", block", block_id,".")
            blocchiFinaliRecuperati = len(self.block_true) + len(self.block_false_saved)
            print("In node", node.originalName, "I managed to recover", blocchiFinaliRecuperati, "blocks.")
```

In case of a Peer-to-Peer system, the assignment was tested with the following parameters in the configuration file:

```
[peer]
number = 10
n = 10
k = 8
data_size = 1 GiB
storage_size = 10 GiB
upload_speed = 2 MiB  # per second
download_speed = 10 MiB  # per second
average_uptime = 8 hours
average_downtime = 16 hours
average_recover_time = 3 days
average_lifetime = 1 year
arrival_time = 0
nodeBad = peer-0,peer-1
fakeName = /
```

These are the results that I have obtained:

- **NODE: peer-0** *local_blocks* [True, True, True, True, True, True, True, True, True, True]

  I am saving, for node peer-1, block 5.

  In node peer-0 I managed to recover 10 blocks.

- **NODE: peer-1** *local_blocks* [True, True, True, True, True, True, True, True, True, True]

  I am saving, for node peer-0, block 0.

  In node peer-1 I managed to recover 10 blocks.

- **NODE: peer-2** *local_blocks* [False, False, False, False, True, False, False, False, False, False]

  I am saving, for node peer-7, block 1.

  I am saving, for node peer-5, block 2.

  I am saving, for node peer-9, block 2.

  I am saving, for node peer-4, block 4.

  I am saving, for node peer-8, block 1.

  In node peer-2 I managed to recover 1 blocks.

- **NODE: peer-3** *local_blocks* [True, False, False, True, False, False, False, False, False, False]

  In node peer-3 I managed to recover 2 blocks.

- **NODE: peer-4** *local_blocks* [False, True, False, False, True, False, False, False, False, False]

    I am saving, for node peer-8, block 4.

    I am saving, for node peer-5, block 0.

    I am saving, for node peer-6, block 2.

  In node peer-4 I managed to recover 2 blocks.
- **NODE: peer-5** *local_blocks* [True, False, True, False, False, False, True, False, False, False]

    I am saving, for node peer-7, block 0.

  In node peer-5 I managed to recover 3 blocks.

- **NODE: peer-6** *local_blocks* [True, False, True, False, True, True, False, False, False, False]

  In node peer-6 I managed to recover 4 blocks.

- **NODE: peer-7** *local_blocks* [True, True, False, True, False, False, False, False, False, False]

    I am saving, for node peer-6, block 0.

  In node peer-7 I managed to recover 3 blocks.

- **NODE: peer-8** *local_blocks* [False, True, False, False, True, True, False, False, False, False]

    I am saving, for node peer-9, block 5.

    I am saving, for node peer-6, block 5.

    I am saving, for node peer-7, block 3.

    I am saving, for node peer-5, block 6.

    I am saving, for node peer-2, block 4.

    I am saving, for node peer-3, block 0.

  In node peer-8 I managed to recover 3 blocks.

- **NODE: peer-9** *local_blocks* [False, False, True, False, False, True, False, False, False, False]

    I am saving, for node peer-6, block 4.

    I am saving, for node peer-8, block 5.

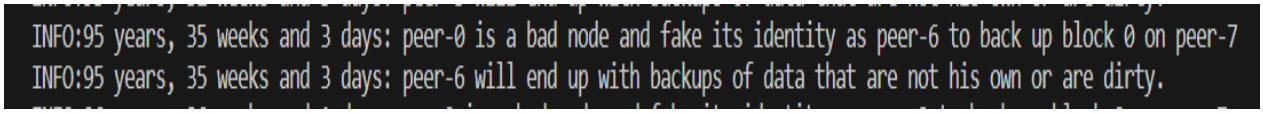
    I am saving, for node peer-3, block 3.

    I am saving, for node peer-4, block 1.

  In node peer-9 I managed to recover 2 blocks.

The malicious peers (peer-0 and peer-1) only contain blocks from the other malicious node in their remote_blocks_held, while the good nodes seemingly have only blocks from other good nodes.

However, by observing the simulation logs, one can notice, for example, that:



This implies that the block saved on peer-7 is not from peer-6 but from peer-0.

In case of a Client-Server system, the assignment was tested with the following parameters in the configuration file:

```
[DEFAULT]  # parameters that are the same by default, for all classes
average_lifetime = 1 year
arrival_time = 0

[client]
number = 3
n = 10
k = 8
data_size = 1 GiB
storage_size = 2 GiB
upload_speed = 500 KiB  # per second
download_speed = 2 MiB  # per second
average_uptime = 8 hours
average_downtime = 16 hours
average_recover_time = 3 days
nodeBad = client-0
fakeName = /

[server]
number = 10
n = 0
k = 0
data_size = 0 GiB
storage_size = 1 TiB
upload_speed = 100 MiB
download_speed = 100 MiB
average_uptime = 30 days
average_downtime = 2 hours
average_recover_time = 1 day
nodeBad = /
fakeName = /
```

These are the results that I have obtained:

- **NODE: client-0** *local_blocks* [True, True, True, True, True, True, True, True, True, True]

  In node client-0 I managed to recover 10 blocks.

- **NODE: client-1** *local_blocks* [True, True, True, True, True, True, True, True, True, True]

  In node client-1 I managed to recover 10 blocks.

- **NODE: client-2** *local_blocks* [True, True, True, True, True, True, True, True, True, True]

  I am saving, for node client-1, block 3.

  In node client-2 I managed to recover 10 blocks.

- **NODE: server-0** local_blocks []

  I am saving, for node client-1, block 0.

  I am saving, for node client-2, block 5.

  In node server-0 I managed to recover 0 blocks.

- **NODE: server-1** *local_blocks* []

  I am saving, for node client-1, block 2.

  I am saving, for node client-2, block 2.

  In node server-1 I managed to recover 0 blocks.

- **NODE: server-2** *local_blocks* []

  I am saving, for node client-2, block 1.

  I am saving, for node client-1, block 4.

  In node server-2 I managed to recover 0 blocks.

- **NODE: server-3** *local_blocks* []

  I am saving, for node client-2, block 3.

  I am saving, for node client-1, block 6.

  In node server-3 I managed to recover 0 blocks.

- **NODE: server-4** *local_blocks* []

  I am saving, for node client-1, block 1.

  I am saving, for node client-2, block 9.

  I am saving, for node client-2, block 7.

  In node server-4 I managed to recover 0 blocks.

- **NODE: server-5** *local_blocks* []

I am saving, for node client-1, block 5.

I am saving, for node client-2, block 0.

In node server-5 I managed to recover 0 blocks.

- **NODE: server-6** *local_blocks* []

  I am saving, for node client-2, block 8.

  I am saving, for node client-1, block 7.

  In node server-6 I managed to recover 0 blocks.

- **NODE: server-7** *local_blocks* []

  I am saving, for node client-1, block 9.

  I am saving, for node client-2, block 6.

  In node server-7 I managed to recover 0 blocks.

- **NODE: server-8** *local_blocks* []

  I am saving, for node client-2, block 4.

  In node server-8 I managed to recover 0 blocks.

- **NODE: server-9** local_blocks []

  I am saving, for node client-1, block 8.

  I am saving, for node client-2, block 7.

  In node server-9 I managed to recover 0 blocks.

In this case, client-0 is the malicious node that can impersonate client-1 and client-2 to save fake blocks on other nodes.

By examining the simulation logs, it can be observed that:

```
INFO:99 years, 24 weeks and 19 hours: client-0 is a bad node and fake its identity as client-2 to back up block 7 on server-9
INFO:99 years, 24 weeks and 19 hours: client-2 will end up with backups of data that are not his own or are dirty.
INFO:99 years, 46 weeks and 2 days: client-0 is a bad node and fake its identity as client-2 to back up block 7 on server-4
INFO:99 years, 46 weeks and 2 days: client-2 will end up with backups of data that are not his own or are dirty.
```

This implies that client-2 will find block 7 saved on two different servers (server-9 and server-4), both of which do not belong to client-2. These servers may contain false or corrupted data.

The obtained data aligns with what I expected – the malicious nodes can save their blocks by pretending to be other nodes. Consequently, if a good node fails and later comes back online, when checking for its saved data on other nodes, it might find blocks that actually do not belong to it.