Distributed Systems
Autumn Semester 2020
Dr. Valerio Schiavoni, Prof. Peter Kropf
Assistant: Isabelly Rocha

Project 2 — Part I
Performance evaluation of distributed systems with THUNDERSTORM

Part 1 Due: 23 November 2020, 23:59

1 Introduction

The performance of distributed systems is heavily affected by the properties of the underlying network. In this project, we will deploy and study how simple and small changes in the network affect the performance of a system.

The project is organized in two parts. The first one introduces you to THUNDERSTORM and will get you familiar with this environment. In the second part, we will study more in depth Redis*, an efficient in-memory key-value store, as well as a few benchmarking tools to evaluate its performance. While it is not the purpose of this project to become experts with Redis, it is recommended (and to some degree, required) to go through its online documentation†, to better understand some of its specificities.

2 Required Reading

For this project (both parts), we will rely on THUNDERSTORM [1], a tool to easily evaluate the performance of distributed systems, including under dynamic network conditions. The paper describing this system is available on ILIAS, as well as at the following address:
`https://libra.unine.ch/export/DL/42530.pdf`
Note that the content of the paper *can be* subject of questions during the exam. Hence, you are highly encouraged to ask questions regarding this system via email, Slack or during the live WebEx sessions. Please always address your questions to all the instructors and the assistant.

3 Requirements

This project should be executed and completed on the dedicated server machine that each of you was provided on our cluster. If you have not received instructions to access it, please contact immediately the instructors and the assistant. This work is individual, *i.e.*, no pairs, no groups.

*`https://redis.io/`
†`https://redis.io/topics/introduction`

Task 0 - Installation (optional - use the server if you can)

Using the server machine on the UniNE cluster will provide you with a fully working environment to complete this project (both parts) and the next one. For your curiosity, on ILIAS you will find an installation script that you can try to use to install THUNDERSTORM and the required libraries and runtime on your Linux machine, the only supported kernel. The script only works on Ubuntu 20.04 LTS. It does not work on Windows. It does not work on MacOS.

Note as well that we do not provide any assistance to help with this installation on your machine. We will not grant any deadline extension due to problems related to installations on your machine. However, if you manage use this script, you are free to proceed and work on your machine. Additionally, if you manage to use it with a different Linux distribution other than Ubuntu 20.04LTS, we will be happy to hear from you.

Task 1 - Getting familiar with THUNDERSTORM

You must be able to SSH into your server and enter into the `thunderstorm` directory. From a Linux-based system, issue the following command with the appropriate values for username and machine identifier:

```
ssh [your-user-name]@[MACHINE_ID].maas
cp -r shared_public/thunderstorm/ thunderstorm/
cd thunderstorm/
ls
```

The `thunderstorm/` directory contains, among others, the following files:

```
example_topology.xml
deploymentGenerator_227
README.md
images/
```

Inspect the `example_topology.xml`. As the name says, this file describes an example network topology that you can use to get familiar with the tool. The topology also describes the applications deployed on top of this topology, as identified by the values of the `image` fields. Such values must be valid names for Docker images [‡], available either on a local registry or on a public one.[§]

Listing 1: Content of example_topology.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<experiment boot="kollaps:1.0">
  <services>
```

[‡] https://docs.docker.com/get-started/
[§] https://hub.docker.com/

```
4       <service name="dashboard"
5         image="kollaps/dashboard:1.0" supervisor="true" port="8088"/>
6       <service name="logger"
7         image="kollaps/logger:1.0" supervisor="true" />
8       <service name="client1"
9         image="kollaps/iperf3-client:1.0" command="['server', '0', '0']"/>
10      <service name="client2"
11        image="kollaps/iperf3-client:1.0" command="['server', '0', '0']"/>
12      <service name="server"
13        image="kollaps/iperf3-server:1.0" share="false"/>
14    </services>
15    <bridges>
16      <bridge name="s1"/>
17    </bridges>
18    <links>
19      <link origin="client1" dest="s1" latency="5"
20            upload="10Mbps" download="10Mbps" network="test_overlay"/>
21      <link origin="client2" dest="s1" latency="5"
22            upload="10Mbps" download="10Mbps" network="test_overlay"/>
23      <link origin="s1" dest="server" latency="5"
24            upload="10Mbps" download="10Mbps" network="test_overlay"/>
25    </links>
26    <dynamic>
27      <schedule name="client1" time="0.0" action="join"/>
28      <schedule name="client2" time="0.0" action="join"/>
29      <schedule name="server"  time="0.0" action="join"/>
30      <schedule name="client1" time="120.0" action="crash"/>
31      <schedule name="client2" time="120.0" action="crash"/>
32      <schedule name="server"  time="120.0" action="leave"/>
33    </dynamic>
34  </experiment>
```

The example deploys three application services: `client1`, `client2` and `server`. Additionally, the example deploys two internal services (`dashboard` and `logger`), which can be ignored. The topology consists of 1 bridge node (`s1`) that connects on one side the clients and on the other side the server. The connections and the properties of the connections are described by the `<links>` elements.

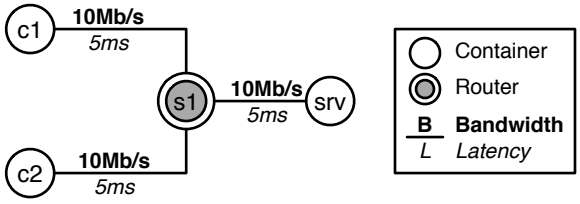

Figure 1: Example topology: 2 clients (c1 and c2), 1 bridge (s1), 1 server (srv).

Figure 1 depicts the example topology. Note that the applications deployed on each node are not represented in the figure, only the corresponding node names are.

The `<dynamic>` block describes events that change the topology. In Listing 1 lines 27-29, the two clients and the server join the system at time 0.0. Instead, at lines 30-31, the two clients are crashed

(SIGKILL) at time 120.0, while the server leaves gracefully (SIGINT).

Task 2 - Topology Deployment

To deploy the topology, we need first to create a *deployment* descriptor. We do this using the deploymentGenerator_227 tool, which you find in the thunderstorm/ directory:

Listing 2: Generation of a deployment descriptor

```
./deploymentGenerator_227 example_topology.xml -s > example_topology.yaml
Graph has 6 links.
      has 5 hosts.
Experiment UUID: 0b46f0dc-bec7-4bfe-9a56-a75ebcf68b91
```

Inspect the content of the generated file example_topology.yaml to get familiar with some of its elements. With this descriptor, it is now possible to deploy the topology:

Listing 3: Start of a deployment

```
docker stack deploy -c example_topology.yaml ThunderStorm
Creating network ThunderStorm_outside
Creating service ThunderStorm_client1-0b46f0dc-bec7-4bfe-9a56-a75ebcf68b91
Creating service ThunderStorm_client2-0b46f0dc-bec7-4bfe-9a56-a75ebcf68b91
Creating service ThunderStorm_server-0b46f0dc-bec7-4bfe-9a56-a75ebcf68b91
Creating service ThunderStorm_bootstrapper
Creating service ThunderStorm_dashboard-0b46f0dc-bec7-4bfe-9a56-a75ebcf68b91
Creating service ThunderStorm_logger-0b46f0dc-bec7-4bfe-9a56-a75ebcf68b91
```

NB: the specific names might differ in your deployment.

As you can see, we rely on Docker [¶] to deploy this example topology. During this process, several Docker containers are created, linked inside an emulated network and ready to be used. Further details about the internal mechanisms are described in [1].

A deployed experiment can be stopped with:

Listing 4: Stop of a deployment

```
docker stack rm ThunderStorm
```

Task 3 - Network measurements

Now, we check the connectivity properties between client1 and server. To do so, we enter into the client1 container by creating a new Bash shell interpreter inside the container:

[¶] https://www.docker.com/

Listing 5: Attach to Bash shell inside a running node

```
1 docker exec -it ThunderStorm_client1-0b46f0dc-bec7-.... /bin/bash
2 bash-4.4#
```

Note that you must provide the full container identifier to the Docker `exec`[‖] command (*NB*: your Bash shell supports auto-completion).

First, from inside `client1`, we want to check the network latency (round-trip time) toward the `server` node. We can do this with the following:

Listing 6: Check ICMP packets latency between `client1` and `server`

```
1 bash-4.4# ping `host server-$NEED_UUID | cut -d ' ' -f 4`
2 PING 10.1.0.119 (10.1.0.119): 56 data bytes
3 64 bytes from 10.1.0.119: seq=0 ttl=64 time=20.316 ms
4 64 bytes from 10.1.0.119: seq=1 ttl=64 time=20.182 ms
5 64 bytes from 10.1.0.119: seq=2 ttl=64 time=20.161 ms
```

The command `ping` takes one parameter, *e.g.* the IP address of the node to ping. This address is extracted by the bash script that follows the ping command (but if you know it beforehand, you can use it directy).

Finally, to check the available bandwidth between `client1` and the *server* without any interference from other nodes, we can do the following.

First, we attach a **new shell** to the server container and start an `iPerf`[**] server:

Listing 7: Attach to Bash shell to the server and starts the iPerf server

```
1 docker exec -it ThunderStorm_server-... /bin/bash
2 bash-4.4# iperf3 -s
3 -----------------------------------------------------------
4 Server listening on 5201
5 -----------------------------------------------------------
```

The commands start the TCP server, which listens for new connections. Next, we connect to `client1` and launch the `iPerf` client:

Listing 8: Attach to Bash shell to the server and starts the iPerf server

```
1 bash-4.4# iperf3 -c `host server-$NEED_UUID | cut -d ' ' -f 4`
2 Connecting to host 10.1.0.119, port 5201
3 [  5] local 10.1.0.126 port 50214 connected to 10.1.0.119 port 5201
```

[‖] https://docs.docker.com/engine/reference/commandline/exec/
[**] https://github.com/esnet/iperf/

```
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  1.44 MBytes  12.1 Mbits/sec    0    132 KBytes
[  5]   1.00-2.00   sec  1.17 MBytes  9.77 Mbits/sec    0    190 KBytes
[  5]   2.00-3.00   sec  1.29 MBytes  10.8 Mbits/sec    0    248 KBytes
[  5]   3.00-4.00   sec  1.29 MBytes  10.8 Mbits/sec    0    307 KBytes
[  5]   4.00-5.00   sec  1.17 MBytes  9.77 Mbits/sec    0    366 KBytes
[  5]   5.00-6.00   sec  1.35 MBytes  11.3 Mbits/sec    0    423 KBytes
[  5]   6.00-7.00   sec  1.53 MBytes  12.9 Mbits/sec    0    482 KBytes
[  5]   7.00-8.00   sec  1.17 MBytes  9.77 Mbits/sec    0    541 KBytes
[  5]   8.00-9.00   sec  1.29 MBytes  10.8 Mbits/sec    0    598 KBytes
[  5]   9.00-10.00  sec  1.41 MBytes  11.8 Mbits/sec    0    657 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec  13.1 MBytes  11.0 Mbits/sec    0             sender
[  5]   0.00-10.58  sec  12.0 MBytes  9.51 Mbits/sec                  receiver

iperf Done.
```

Note that the server reports similar results.

Task 4 - Simple point-to-point topology, `ping` and `iperf`

Adapt the example topology to have a simple point-to-point topology. In particular, the topology must only include: one client, one server, and two switches (`s1` and `s2`) chained together, and sitting between the client and the server nodes. The link between the `client` and `s1` must have a 100Mb/s bandwidth and 10ms link-latency. The link between the `server` and `s2` has the same characteristics. Finally, the link between the two switches has a bandwidth of 200Mb/s and a 5ms latency.

Expected Task 4 results. For this task, you must report:

- the .xml file describing this topology
- the corresponding yaml file
- the command used and the output of `ping` and `iperf` measured from the client toward the server. What is the average latency ?

Task 5 - Bandwidth sharing

Using the example topology (hence, with two clients), deploy and start at the the same time an `iPerf` benchmark on the two clients. Note that each iPerf server can only handle requests from one client, so you will need to instantiate another iPerf server on the same node.

Expected Task 5 results. For this task, you must report:

- the command and the output of `ping` and `iperf`, from both clients toward the server.
- a plot representing the throughput of each client (one single plot, two curves). On the x-axis the plot reports the time of the experiment, on the y-axis the reported throughput. The experiment should last at least 60 seconds. What do you observe?

Task 6 - Redis deployment

In this task, we will deploy Redis on top of the simple point-to-point topology done in Task 4. To overcome some of the security restrictions enabled by the default in Redis, we will build a custom Redis image that we can use in our experiments. In the `thunderstorm/images/` directory, create a new `redis` directory, and create a new file called `Dockerfile`. Its content looks like this:

Listing 9: File `Dockerfile` for Redis

```
1 FROM redis
2 COPY redis.conf /usr/local/etc/redis/redis.conf
3 CMD [ "redis-server", "/usr/local/etc/redis/redis.conf" ]
```

This Dockerfile will use a custom `redis.conf` configuration file. This `redis.conf` can be retrieved from: `https://gist.githubusercontent.com/vschiavoni/334dc587d96f301ec11f46b212f64984/raw/585319ce74fd7d720b2aa330e3657ee42fec78cb/redis.conf` and also available on ILIAS.

These two files must be in the same directory `images/redis/`. From there, build the new image:

Listing 10: Building the Redis image

```
1  $docker build -t kollaps/redis:1.0 .
2  Sending build context to Docker daemon  64.51kB
3  Step 1/3 : FROM redis
4   ---> de25a81a5a0b
5  Step 2/3 : COPY redis.conf /usr/local/etc/redis/redis.conf
6   ---> Using cache
7   ---> b63b1e6c8ec9
8  Step 3/3 : CMD [ "redis-server", "/usr/local/etc/redis/redis.conf" ]
9   ---> Using cache
10  ---> d8876c667f71
11 Successfully built d8876c667f71
12 Successfully tagged kollaps/redis:1.0
```

Once complete, this image can be used from inside topology descriptor files with the name `kollaps/redis:1.0`. Modify the topology in Task 4 to use `kollaps/redis:1.0` instead of the current one.

Expected Task 6 results.

- the .xml file describing this topology
- the generated .yaml file
- the output from the `docker stack ...` command used to deploy the topology.

Task 7 - Redis: start and functional tests

At this point, the Redis server container has been deployed. Yet, the service itself is not running. This is because in THUNDERSTORM , the services must be started explicitly. The goal of this task is to start the Redis container and check that a simple command-line client for Redis can connect to it. In this task, we will first attach to one of the other *internal* containers, listed upon deployment of the topology, *i.e.*, the `dashboard` container, which is in charge of starting the services available in this experiment.

Listing 11: Starting the Redis service

```
1 docker exec -it ThunderStorm_dashboard-... /bin/bash
2 #apk --update add curl
3 #curl 127.0.0.1:8088/start
4 [some redirecting message here, this is expected]
```

As you can infer, the `dashboard` container exposes a REST-based interface to manage the experiment, *e.g.*, to start and stop it (plus other operations not relevant for us). We query this interface using the `curl` tool. Once the services are started, we can connect to it. To do so, we use the `client1` container, install a simple command-line tool for Redis and connect to it:

Listing 12: Redis-CLI: connection

```
1  docker exec -it ThunderStorm_client1-.. /bin/bash
2  bash-4.4# apk --update add redis
3  fetch http://dl-cdn.alpinelinux.org/alpine/v3.9/main/x86_64/APKINDEX.tar.gz
4  fetch http://dl-cdn.alpinelinux.org/alpine/v3.9/community/x86_64/APKINDEX.tar.gz
5  (1/1) Installing redis (4.0.14-r0)
6  Executing redis-4.0.14-r0.pre-install
7  Executing redis-4.0.14-r0.post-install
8  Executing busybox-1.29.3-r10.trigger
9  OK: 24 MiB in 33 packages
10 bash-4.4# redis-cli -h ThunderStorm_redis-$NEED_UUID
11 ThunderStorm_redis-9f865e23-a003-4508-8cfd-4d8be706fb95:6379>
```

If you reach this point, good job! Redis is now deployed on top of the simple point-to-point topology and can be used via command-line tools or other benchmarking software. You can test a simple operation (setting and getting the value of a variable):

Listing 13: Redis-CLI: example

```
ThunderStorm_redis-9f865e23-a003-4508-8cfd-4d8be706fb95:6379[2]> 5 incr
    mycounter
(integer) 1
(integer) 2
(integer) 3
(integer) 4
(integer) 5
ThunderStorm_redis-9f865e23-a003-4508-8cfd-4d8be706fb95:6379[2]> get mycounter
"5"
```

You can read about other supported commands from the online documentation.[††]

Expected Task 7 results.

- the .xml file describing this topology
- the generated .yaml file
- the output from the `docker stack ...` command used to deploy the topology.

Submission

The work is individual. No exceptions. The following documents must be uploaded on ILIAS:

1. A pdf version of your report. Each student has to implement his/her own version of the report. This report should contain all your interesting results (in table and/or plot formats), as well as a pertinent analysis and evaluation.
2. A zip file containing all the files you produced (PDF report, xml files, etc.)

Please respect the following point:

- Make sure that your compressed document is of reasonable size (no need for images of high-quality) and upload it on ILIAS by strictly following the naming conventions.

Note that the fact that you strictly followed all the instructions concerning the way to provide your report and results (formatting, type of file, deadline,...) will be taken into account to establish your mark.

[††] `https://redis.io/commands`

References

[1] Luca Liechti, Paulo Gouveia, João Neves, Peter Kropf, Miguel Matos, Valerio Schiavoni. *THUNDERSTORM: a tool to evaluate dynamic network topologies on distributed systems*. Proceedings of IEEE SRDS 2019 (38^{th} IEEE International Symposium on Reliable Distributed Systems). Lyon, France, October 2019.