Assignment 1 Part A

INF-3200 Distributed Systems Fundamentals

Throughout this semester, you will complete assignments to build a functional and efficient peer-to-peer distributed hash table. This system should be implemented in a manner that supports real physical distribution of processes across heterogeneous hardware. To achieve this, the system will be implemented on a computer cluster that you will be provided access to through the UiT Department of Computer Science.

The system will be implemented from scratch, and you are allowed to use whatever programming language you prefer. The cluster environment has support for Python, Go, C, and Rust. If you want to use a language, version, or library that is not installed on the cluster, you can install it locally on your own user.

Because you will implement a somewhat large-scale system in this new programming environment, we have decided to split the first assignment in two parts, A and B. In this part A, you will get used to working on the compute cluster and producing code and results that can be reproduced by other users. The details about the protocol and properties of the hash table that you will implement, will be discussed in part B. In short, later assignments will have you implement servers that communicate with each other to store and retrieve data from deterministic locations, through multiple hops.

Implementation and pre-code

In this assignment, you will implement the following:

1. A simple server that is hosted on a given port and responds to an HTTP request, implemented in the programming language and with the libraries of your choosing.
   - **API**: HTTP GET requests to `/helloworld` should respond with a string containing the server's own host-port combo. For example, with a server running at c0-1:50153, the request `http://c0-1:50153/helloworld` should receive the response `"c0-1:50153"`.
2. A shell-script named `run.sh` that deploys several such servers on several nodes on the cluster. **The servers should automatically start at different nodes within the compute cluster**. Only when the requested number of servers exceeds the number of different nodes in the cluster should two servers be launched on the same node. You can choose to implement this as a separate script in a different language, and launch that script via the `run.sh`-script if you desire, as long as the other constraints are satisfied.
   - **Input**: The run-script should take an integer as an input parameter; this number signifies how many servers the script should start.
   - **Output**: The run-script should output the host-port combo of the running servers, formatted as a json-list, for example "["c0-1:50153", "c1-0:49001", "c1-1:55737"]"

You will be provided with the following pre-code:

1. A Python script `testscript.py` that takes an input parameter in the form of a list of host-port combos, such as in the example above, and tries to communicate with the servers with the HTTP GET request `/helloworld`. This file should not be modified except for testing purposes.

The desired output of a successfull solution to this assignment is a `run.sh` that automatically deploys any number of servers to the cluster, and a `testscript.py` that receives information from all servers.

Here is an example run:

```
./run.sh 3
["c0-1:50153", "c1-0:49001", "c1-1:55737"]

python3 testscript.py '["c0-1:50153", "c1-0:49001", "c1-1:55737"]'
received "c0-1:50153"
received "c1-0:49001"
received "c1-1:55737"
Success!
```

Important things to note:

1. Following the formats specified above is a key requirement in this assignment. Not implementing the API correctly will result in a failed assignment. If you are in doubt, look at `testscript.py` and match the outputs that it is expecting.
2. **Running the script `run.sh` for the first time should deploy the system without the need for any other manual intervention or file path modifications**. This means that any dependencies that your system requires, such as libraries, packages, or different compiler versions, should be resolved automatically by running this script. This restriction will be in place for all assignments in this course, and is an exercise in reproducibility, and by extent code mobility.
3. Your server processes must run on different nodes on the cluster. This means that your code should be able to find available hosts and port numbers, and automatically deploy servers accordingly.

Deliverables

The server code, the run-script, and any other files you may have produced, should be delivered in a zip-file named after your UiT username (typically three letters and three numbers, or five letters and four numbers, for example *aov014* or *arove0014*). **Within the zip-file, the top-level folder should also be named after your UiT username.** This somewhat stringent naming scheme makes assessment easier and faster for the TAs.

Within the top-level folder, there should be a *src* folder, containing your implementation, and a *doc* folder, containing a plaintext file listing the members of the group who collaborated on this hand-in. There is no requirement of a report in assignment 1 part A.

Example folder structure:

```
aov014.zip/
    aov014/
        src/
```

```
            server.py
            run.sh
        doc/
            group.txt
```

Practical information

- See Canvas for deadline
- You may work in groups of one to three members. In general, a higher quality of work is expected of groups of three.
- Every student must deliver their own hand-in on Canvas, even when working in a group. Every group member is responsible for their own hand-in.

Intro to the IFI Development Cluster

Guide originally written by Mike Murphy and adapted by Aril Ovesen.

Accessing the cluster

Our development cluster is called `ificluster`, and it lives at `ificluster.ifi.uit.no`. Everyone in the class will be given a user account on the cluster. Your username will be the same as your UiT user name, but the password will be different. You should receive an email from "UiT IFI Kontoadmin" with an initial password.

To log in, SSH to `ificluster.ifi.uit.no` (edit this to use your username):

```
$ ssh aov014@ificluster.ifi.uit.no
```

Note that the cluster is only accessible from inside the UiT/Eduroam networks. It is possible to connect to the UiT network through the VPN, which is managed by Orakelet. See their information here.

Using the command Line

The cluster is made up of Linux machines and all access is through the command line. If you are unfamiliar with the Linux command line, the TAs may be able to help you by answering questions, but a full introduction to Linux and the command line are beyond the scope of this course. If you are not comfortable with working through the command line, it is **highly recommended that you take some time to get familiar with it as soon as possible**.

Text Editors

Common command-line text editors like nano, vim, and emacs are installed. Nano is the simplest, barebones option, whereas vim and emacs require more set-up and getting used to. Using either of these for remote work is very handy, and is recommended if you have the time and interest in learning them. However, GUI-based editors also work. Setting up Visual Studio Code to open remote folders over ssh is easy with the remote-ssh extension. Look for similar extensions if you use other text editors. It is also possible to mount the cluster's file system to a local directory, but using this for live code editing may be slow and unreliable, and the above methods are recommended instead.

Cluster Architecture

When logging on to the cluster, you should be greeted with the following message:

```
All work should be carried out on the compute nodes
List nodes available right now: /share/ifi/available-nodes.sh
List the cluster hardware: /share/ifi/list-cluster-static.sh
List nodes by load: /share/ifi/list-nodes-by-load.sh
List all your processes: /share/ifi/list-cluster-my-processes.sh
Clean out all your processes: /share/ifi/cleanup.sh
Check your disk use: du -hs $HOME
Technical support email: tk@cs.uit.no
```

These scripts are useful for navigating the cluster.

The cluster consists of a “front-end” machine, simply named `ificluster` that faces the outside world, and several “compute” machines behind it, with names on the format `cX-Y`

You can see all the hosts and their hardware by executing `/share/ifi/list-cluster-static.sh`

```
[aov014@ificluster ~]$ /share/ifi/list-cluster-static.sh
HOSTNAME, MODEL, # CPUS, # CORES, # LOGICAL PROCESSORS, PROCESSOR TYPE, RAM, NETWORK SWITCH,
GRAPHICS, DISK

ificluster, Dell PE R440, CPU 1, cores 10, processors 10, Xeon Silver 4114 @ 2.20GHz, RAM
64GB (8 x 8GB), sw, Matrox, RAID

    c0-0, HP DL360p Gen8, CPU 2, cores 8, processors 16, Xeon E5-2670 0 @ 2.60GHz, RAM 192GB
(24 x 8GB), sw2, Matrox,
    c0-1, HP DL360p Gen8, CPU 2, cores 8, processors 16, Xeon E5-2670 0 @ 2.60GHz, RAM 192GB
(24 x 8GB), sw2, Matrox,
    c1-0, Dell PE R420, CPU 2, cores 6, processors 12, Xeon E5-2430 v2 @ 2.50GHz, RAM 64GB
(8 x 8GB), sw2, Matrox, SAS 146 GB
    c1-1, Dell PE R420, CPU 2, cores 6, processors 12, Xeon E5-2430 v2 @ 2.50GHz, RAM 64GB
(8 x 8GB), sw2, Matrox, SAS 146 GB
...
```

To see the nodes that are available right now, run `/share/ifi/available-nodes.sh` (might be slow):

```
[aov014@ificluster ~]$ /share/ifi/available-nodes.sh
c0-0
c0-1
c1-0
c1-1
...
```

When you first log in, you will be on the `ificluster` front-end. From there, you can SSH to any compute node without a password.

```
[aov014@ificluster ~]$ echo $HOSTNAME
ificluster.ifi.uit.no
[aov014@ificluster ~]$ ssh c7-23
[aov014@c7-23 ~]$ echo $HOSTNAME
c7-23.ifi.uit.no
```

To avoid overloading the front-end machine, you are required to do all your actual work on the compute nodes. ***It is important that you do not execute your own scripts and test your assignment implementation on the front-end node.***

All of these computers share an NFS `/home/` partition. So any files you have in your home directory on the front-end will be present on all compute nodes in the same place. This means that when you're working on assignments, you don't have to worry about updating and compiling code on every machine.

Launching a Background Process on a Compute Node

To launch a program in the background on a cluster machine, use SSH's `-f` flag.

```
[aov014@c7-23 ~]$ ssh -f c5-3 "cat /etc/hostname"
c5-3.ifi.uit.no
```

Note that, when you run a command over SSH like this, it will launch from the home directory of the target machine, rather than the subdirectory that you are in locally.

```
$ cd my_project
$ ./project_executable
# (On the local machine you are in the my_project directory)
$ ssh -f compute-1-1 ./project_executable
./project_executable: No such file or directory
# (On the target machine you are back to the home directory,
#   and the executable is not found there)
```

You can pass the current directory to the target machine by specifying it in the command line. The `$PWD` variable is useful for this. It will expand locally, and then be sent to the target machine as part of the command line:

```
$ cd my_project
$ ssh -f c1-1 $PWD/project_executable
# This expands to
# ssh -f c1-1 my_project/project_executable
```

To terminate the process you can use `killall`:

```
# Kill specific executable
$ ssh c1-1 killall project_executable
# Kill any processes that you are running
$ ssh c1-1 killall -u $(id -un)
```

Note that SSH will continue to pipe the programs stdout and stderr back to your terminal. This can be helpful if you want to watch a server's logs, but it can be annoying when you're trying to type more commands in the terminal. You can always redirect the output to a file with the shell's standard redirection features:

```
$ ssh -f c1-1 $PWD/project_executable 2>&1 > output.txt
```

Watch Out for Output Buffering

When running the Python starter code over SSH, you might find that there seems to be no output. This is because the Python interpreter is a little too clever for its own good. It detects that it seems to be in a non-interactive shell and turns on output buffering. The script is still producing output, but you are not seeing it because the output buffer has not been flushed.

If you have this problem, try invoking the code with `python3 -u`.

```
$ ssh c2-1 python3 -u $PWD/dummynode.py
```

Sharing the Cluster

The most important thing to remember about the cluster is that we are all sharing it. So be careful not to hog resources. This is why we require you to do your computation on the compute nodes.

Your user privileges are limited, so you shouldn't have to worry about deleting someone else's files or screwing up the cluster itself. However, network resources like port numbers and open connections are global and finite. If you are not careful with how you use the network you might accidentally DoS yourself and your fellow students.

Don't leave things running

When you are done with your work, be sure to kill all of your processes. We have had instances in the past where a student started an intensive process, turned off his monitor, and then went hiking on Kvaløya. Please do not do this.

The `cleanup.sh` script will kill all of your processes and log you out. Please run it when you are done working.

```
[aov014@ificluster ~]$ /share/ifi/cleanup.sh
Connection to ificluster.ifi.uit.no closed by remote host.
Connection to ificluster.ifi.uit.no closed.
```

Also, we encourage you to write a time limit into your processes and servers so that they will shut down automatically if you forget about them.

Choose Your Own Port Numbers

It is especially easy to get into conflict over network ports. Our assignments will involve running servers on the cluster, and only one server at a time can listen on any given port. So if you and another student are using the same port numbers for your server, you will end up in conflict. If you see "address already in use" errors, you are in conflict over a port. To avoid conflicts, pick port numbers at random for all your servers. The ephemeral port range, 49152 to 65535, is a good place to pick from.

To pick a random port number on the command line, you can use the `shuf` command (shuffle).

```
$ shuf -i 49152-65535 -n1
61249
$ shuf -i 49152-65535 -n1
49631
```

```
$ shuf -i 49152-65535 -n1
50522
```