Remote procedure calls

I) Introduction

gRPC[1] is a Remote Procedure Call system, concurrent of **REST**[2] that enables highly efficient communication in distributed client/server architectures using innovative process technologies. Like its predecessor RPC, it operates at the process level. The characteristic feature of inter-process communication via gRPC is the principle of transparency: the relationship between remote instances is intended to be as close and simple as local communication between several internal processes on the same machine.

The gRPC was developed by Google in 2015. Today, the Cloud Native Computing Foundation is taking over the dissemination and development of this communication technique. The gRPC is open source, which means that its source text is in the public domain, third parties are allowed and even encouraged to modify it and further develop it.

The gRPC handles the transport of data streams between remote computers via the **HTTP/2** protocol in a standardized way. The **Protocol Buffers** developed by Google take care of the structuring and processing of data. These are stored as simple text files with the extension.proto.

The gRPC is often understood as a framework. One of the features of a framework is that it provides a programming framework for its developers, saving time and effort. The standardized structure of gRPC already includes various elements and functions that do not have to be reprogrammed all the time. In addition, the gRPC framework defines standardized interfaces for defined sources.

II) Implementation of a distributed application

2.1 Setting in situation

We work in a research laboratory, and you are responsible for the server rack available for our group. We have several servers, forming a computing cluster. All members of the lab need to perform computing tasks for several hours. So, we want to implement a manager that allows clients to submit their task in a FIFO queue and decide which server should perform the task. A server can only perform one task at a time. That is why we must use gRPC.

2.2 Settlement

[1] **gRPC :** google remote procedure calls
[2] **REST :** architectural style and a communication mode frequently used in the development of Web services

We have read the entire work to understand what to do and how. We are now ready to develop an RPC program.

2.3 Development of a RPC program

```
//TODO : send the gRPC call
    Status status = this->stub_->ProcessOperation(&context, request, &reply);
```

To send a call to the gRPC we call the same method as the server, of **ProcessOperation**[3].

```
//TODO : Defines the service, the request and Reply
service Operation {
  rpc ProcessOperation (OperationRequest) returns (OpreationReply) {}
}
message OperationRequest {
  string task = 1;
}
message OperationReply {
  string message = 1;
}
```

For the proto file, we added the two messages of Request and reply with both variable tasks found in the manager file and message in the server file.

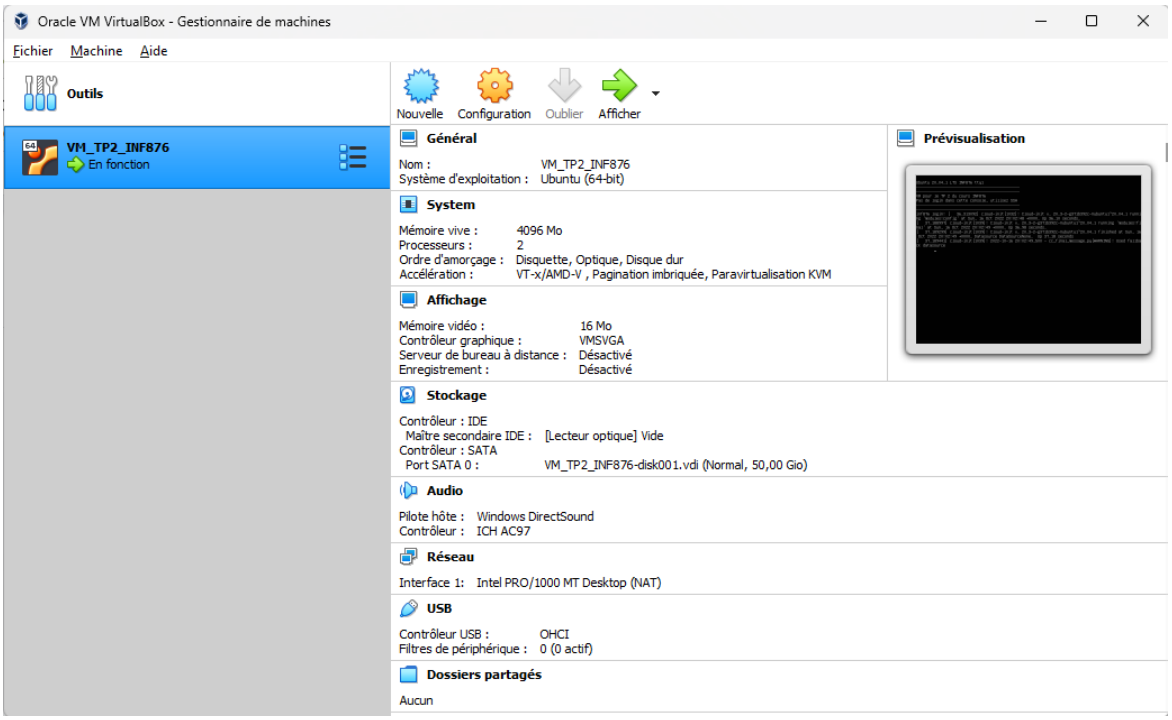
```
std::string task(argv[1]);
```

To take the task as a command line parameter we get the argv variable.

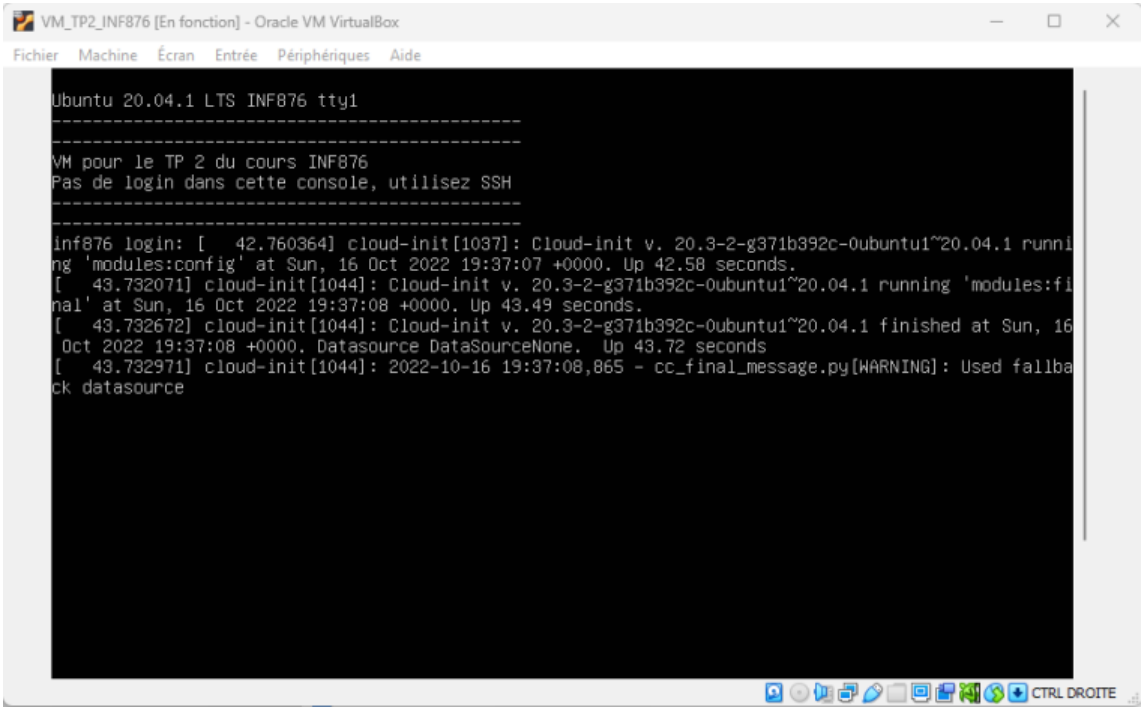
2.4.1 Deploying the virtual machine on the PC

Once the virtual image has been downloaded to our machine from the **Moodle** link, we can import it directly into our **Oracle VM**[4] **VirtualBox** software.

[3] **Process Operation :** the mass production method of producing products in a continuous flow.
[4] **VM :** Virtual Machine



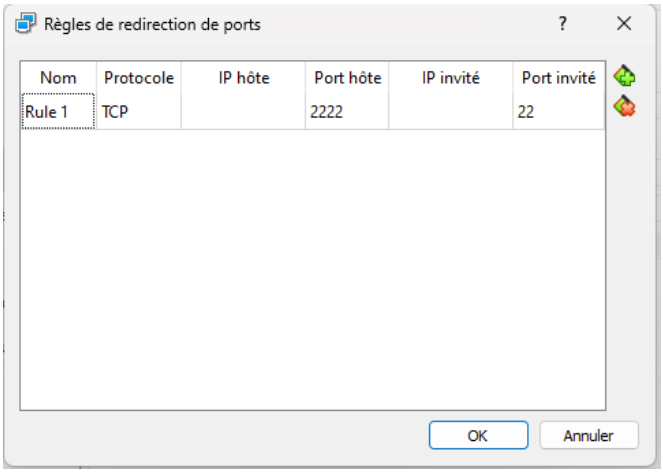
The image runs perfectly on our 8 GB RAM machine as indicated on the statement.



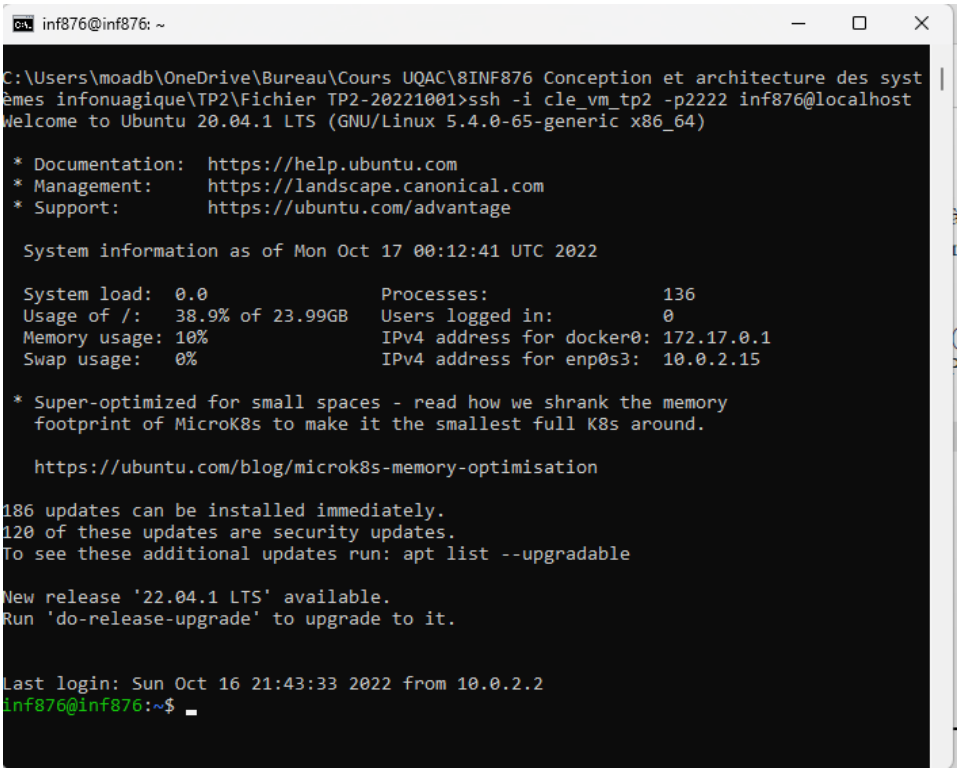
We also provide the file "cle_vm_tp2" which will allow to launch the virtual machine without having to authenticate.

Logs	16/10/2022 15:58	Dossier de fichiers	
cle_vm_tp2	16/10/2022 15:25	Fichier	3 Ko
VM_TP2_INF876	16/10/2022 16:00	VirtualBox Machin	4 Ko

And of course, we configure the port forwarding on the VM VirtualBox software.



This allows us to connect from our physical machine:



2.4 Use of a container

The container already contains the Docker tool, **Protobuf** [5]and the gRPC C++ libraries.
We test Docker if it is well installed with a simple command.

[5] **Protobuf :** serialization format with an interface description language developed by Google

```
Last login: Sun Oct 16 21:43:33 2022 from 10.0.2.2
inf876@inf876:~$ sudo docker ps
[sudo] password for inf876:
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
cd6d63d45668        inf876tp2:v1        "bash"              2 hours ago         Up 2 hours                              inf876tp2
```

We check that the Docker image is present on the VM.

```
Last login: Sun Oct 16 21:43:33 2022 from 10.0.2.2
inf876@inf876:~$ sudo docker ps
[sudo] password for inf876:
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
cd6d63d45668        inf876tp2:v1        "bash"              2 hours ago         Up 2 hours                              inf876tp2
```

We deploy a Docker container based on the above image "inf876tp2:v1".

```
inf876@inf876:~$ sudo docker run -dit --name inf876tp2 inf876tp2:v1
1891ac2db2d9af256525be8d25f3c2b8054b062e8161bedaa1fefa9d0988624d
```

2.5 Tests

To test our deployment, we proceed with the following steps:

- ✓ Upload files to the VM from the physical machine :

```
C:\Users\moadb\OneDrive\Bureau\Cours UQAC\8INF876 Conception et architecture des systèmes infonuagique\TP2\Fichier TP2-2
0221001>scp -i cle_vm_tp2 -P2222 TP2_trace_partie1_v1.zip inf876@localhost:
TP2_trace_partie1_v1.zip                                      100% 6111     1.0MB/s   00:00
```

- ✓ We upload the files to the Docker container from the VM :

```
Copy files/folders between a container and the local filesystem
inf876@inf876:~$ sudo docker cp TP2_trace_partie1_v1.zip inf876tp2:/root
```

- ✓ We connect to the Docker container :

```
inf876@inf876:~$ sudo docker exec -it inf876tp2 /bin/bash
root@cd6d63d45668:/#
```

- ✓ We move to the directory where the files have been transferred :

```
root@cd6d63d45668:/# cd /root/
root@cd6d63d45668:~#
```

- ✓ We compile our trace provider in the folder "lttng-traces" :

```
root@cd6d63d45668:~/TP2_trace_partie1/task_scheduler/lttng-traces# gcc -I. -c grpc_tracing.c
root@cd6d63d45668:~/TP2_trace_partie1/task_scheduler/lttng-traces#
```

✓ We move back one folder and run the "make" command:

```
root@cd6d63d45668:~/TP2_trace_partie1/task_scheduler# make clean
rm -f *.o *.pb.cc *.pb.h manager server
root@cd6d63d45668:~/TP2_trace_partie1/task_scheduler# make
protoc -I . --cpp_out=. operation.proto
g++ -std=c++11 `pkg-config --cflags protobuf grpc`  -c -o operation.pb.o operation.pb.cc
protoc -I . --grpc_out=. --plugin=protoc-gen-grpc=`which grpc_cpp_plugin` operation.proto
g++ -std=c++11 `pkg-config --cflags protobuf grpc`  -c -o operation.grpc.pb.o operation.grpc.pb.cc
g++ -std=c++11 `pkg-config --cflags protobuf grpc`  -c -o manager.o manager.cc
g++ operation.pb.o operation.grpc.pb.o manager.o lttng-traces/grpc_tracing.o -L/usr/local/lib `pkg-config --libs protobuf grpc++ grpc` -Wl,--no-as-needed -lgrpc++_refle
ction -Wl,--as-needed -ldl -llttng-ust -o manager
g++ -std=c++11 `pkg-config --cflags protobuf grpc`  -c -o server.o server.cc
g++ operation.pb.o operation.grpc.pb.o server.o lttng-traces/grpc_tracing.o -L/usr/local/lib `pkg-config --libs protobuf grpc++ grpc` -Wl,--no-as-needed -lgrpc++_reflec
tion -Wl,--as-needed -ldl -llttng-ust -o server
root@cd6d63d45668:~/TP2_trace_partie1/task_scheduler#
```

✓ We launch the **tracing**...

```
root@cd6d63d45668:~/TP2_trace_partie1/task_scheduler# sudo sh trace.sh
Session auto-20221017-003534 created.
Traces will be written in /root/TP2_trace_partie1/task_scheduler/trace_files/1665966934488
UST channel ustchannel enabled for session auto-20221017-003534
All UST events are enabled in channel ustchannel
UST context vpid added to channel ustchannel
UST context vtid added to channel ustchannel
Tracing started for session auto-20221017-003534
root@cd6d63d45668:~/TP2_trace_partie1/task_scheduler#
```

✓ ... and the **server** in the background :

```
root@cd6d63d45668:~/TP2_trace_partie1/task_scheduler# ./server&
[1] 922
root@cd6d63d45668:~/TP2_trace_partie1/task_scheduler# Server listening on 0.0.0.0:50051
```

✓ We launch tasks to be executed :

```
root@cd6d63d45668:~/TP2_trace_partie1/task_scheduler# ./manager tache23
Reply from server : tache23
root@cd6d63d45668:~/TP2_trace_partie1/task_scheduler#
```

```
root@cd6d63d45668:~/TP2_trace_partie1/task_scheduler# ./manager tache45
Reply from server : tache45
```

✓ We can stop the tracing :

```
root@cd6d63d45668:~/TP2_trace_partie1/task_scheduler# lttng destroy
Session auto-20221017-003534 destroyed
root@cd6d63d45668:~/TP2_trace_partie1/task_scheduler# lttng stop
```

✓ We transfer the tracking files from the container to the VM :

```
inf876@inf876:~$ sudo docker exec -it inf876tp2 /bin/bash
root@cd6d63d45668:/# cd /root/
root@cd6d63d45668:~# ls
TP2_trace_partie1  TP2_trace_partie1_v1.zip
root@cd6d63d45668:~# exit
exit
inf876@inf876:~$ sudo docker cp cd6d63d45668:/root/TP2_trace_partie1/task_scheduler/trace_files/ .
inf876@inf876:~$ ls
TP2_trace_partie1.tar.gz  correction_tp2.sh      nohup.out                     trace_files
TP2_trace_partie1_v1.zip  correction_tp2.sh.x    task_scheduler_correction     trace_files.zip
cle_vm_tp2                correction_tp2.sh.x.c  task_scheduler_correction.zip  trace_files2.zip
inf876@inf876:~$
```

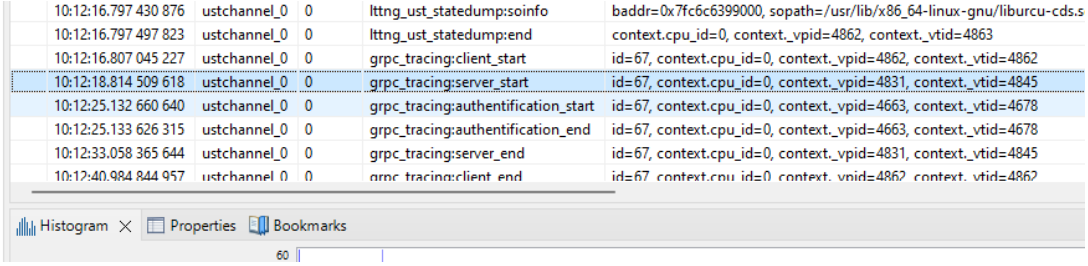
- ✓ We zip our files in a single archive :

```
inf876@inf876:~$ sudo zip -r trace_files3.zip trace_files
  adding: trace_files/ (stored 0%)
  adding: trace_files/1665958676999/ (stored 0%)
  adding: trace_files/1665958676999/ust/ (stored 0%)
  adding: trace_files/1665958676999/ust/uid/ (stored 0%)
  adding: trace_files/1665958676999/ust/uid/0/ (stored 0%)
  adding: trace_files/1665958676999/ust/uid/0/64-bit/ (stored 0%)
  adding: trace_files/1665958676999/ust/uid/0/64-bit/metadata (deflated 91%)
  adding: trace_files/1665958676999/ust/uid/0/64-bit/ustchannel_0 (deflated 76%)
  adding: trace_files/1665958676999/ust/uid/0/64-bit/index/ (stored 0%)
  adding: trace_files/1665958676999/ust/uid/0/64-bit/index/ustchannel_1.idx (deflated 51%)
  adding: trace_files/1665958676999/ust/uid/0/64-bit/index/ustchannel_0.idx (deflated 53%)
  adding: trace_files/1665958676999/ust/uid/0/64-bit/ustchannel_1 (deflated 76%)
  adding: trace_files/1665957678049/ (stored 0%)
  adding: trace_files/1665957678049/ust/ (stored 0%)
  adding: trace_files/1665957678049/ust/uid/ (stored 0%)
  adding: trace_files/1665957678049/ust/uid/0/ (stored 0%)
  adding: trace_files/1665957678049/ust/uid/0/64-bit/ (stored 0%)
  adding: trace_files/1665957678049/ust/uid/0/64-bit/metadata (deflated 90%)
  adding: trace_files/1665957678049/ust/uid/0/64-bit/ustchannel_0 (deflated 81%)
  adding: trace_files/1665957678049/ust/uid/0/64-bit/index/ (stored 0%)
  adding: trace_files/1665957678049/ust/uid/0/64-bit/index/ustchannel_1.idx (deflated 51%)
  adding: trace_files/1665957678049/ust/uid/0/64-bit/index/ustchannel_0.idx (deflated 66%)
  adding: trace_files/1665957678049/ust/uid/0/64-bit/ustchannel_1 (deflated 76%)
  adding: trace_files/1665962620079/ (stored 0%)
```

- ✓ We transfer them to the physical machine :

```
C:\Users\moadb\OneDrive\Bureau\Cours UQAC\8INF876 Conception et architecture des systèmes infonuagique\TP2\Fichier TP2-2
0221001>scp -i cle_vm_tp2 -P2222  inf876@localhost:~/trace_files3.zip .
trace_files3.zip                                              100%   67KB   6.1MB/s   00:00
```

- ✓ We open our plotting results on the **Trace Compass** [6]software and we obtain results comparable to the statement :



[6] **Trace Compass :** it is a Java tool for viewing and analyzing any type of logs or traces. Its goal is to provide views, graphs, metrics, etc. to help extract useful information from traces, in a way that is more user-friendly and informative than huge text dumps.

III) Analysis of a system trace of a distributed system

3.1 Setting in situation

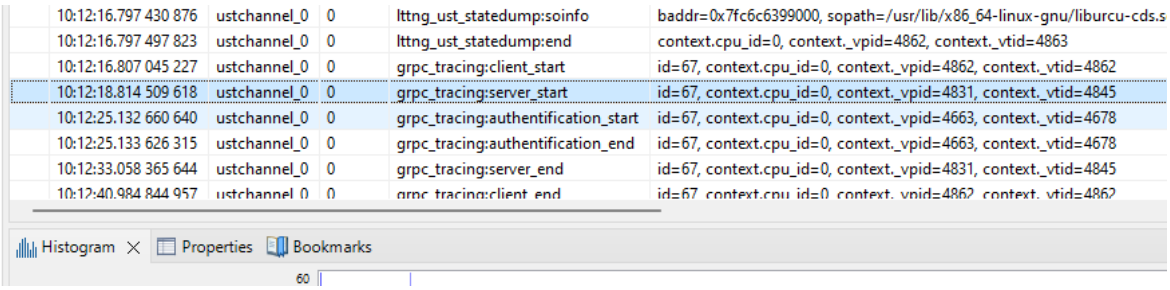
We are an analyst in the IT department at UQAC, and it was reported to us that the human resources software was slow.

The site is a distributed system, composed of three parts (client application, application server and authentication server) which are gRPC applications. When connecting, the client sends a request to the application server, and the application server verifies the identity of the client with the authentication server.

Therefore, we need to find the cause and duration of bottlenecks. Bottlenecks can be in the client, the server, or the authentication server.

3.2 Analysis

We used this **delta** to get the timespan between two operations. The command is "Ctrl + Maj".



Execution of the client :

- The request from client to server : 2.007464391 sec
- The reply from server to client : 7.926479313 sec

Global client timespan : 9.933943704 sec

Execution of the server:

- The request from the server to the auth server : 6.318151022 sec
- The reply from the auth server to the server : 7.924739329 sec

Global server timespan : 14.242890351 sec

3.3 Verification and delivery

We notice that the shortest operation is the connection from the client to the server (client start).

Moreover, the most expensive operation is the response from the server to the client (client end).

Overall, server-side requests take longer than client-side requests (authentication start & end).

Thus, the bottlenecks are found at the server level, mainly concerning the authentication to the server, then concerning the response from the server to the client.

To conclude, this shows a fast and powerful technology, but which requires an authentication time because it is a more robust technology than what its competitor REST offers. In this way, even older programs can be easily extended with a powerful gRPC interface and large files can be transferred much faster.