POLYTECHNIQUE MONTRÉAL

INF8480 - Systèmes répartis et infonuagique

TP2 - Appels de méthodes à distance

Chargés de laboratoire :
Pierre-Frederick Denys

Automne 2020 - V4.1

1 Introduction

1.1 Prérequis

— **Intergiciels et objets répartis, Communication inter-processus, Messages de groupes :** Sun RPC, gRPC, CORBA, Java RMI et .NET Remoting.

1.2 But du TP

— Introduction à gRPC
— Introduction au calcul réparti et à la répartition de tâches
— Analyse de performance des systèmes répartis à l'aide du traçage

Le TP comporte deux parties indépendantes, il est vivement conseillé que les deux membres du binôme travaillent et comprennent les concepts abordés dans les deux parties du TP.

2 Partie 1 : Implémentation d'une application répartie

2.1 Mise en situation

Vous travaillez dans un laboratoire de recherche, et vous êtes responsable de la baie de serveur disponible pour votre groupe. Vous disposez de plusieurs serveurs, formant une grappe de calcul. Tous les membres du laboratoire ont besoin d'effectuer des tâches de calcul durant plusieurs heures.

Vous souhaitez donc implémenter un gestionnaire qui permet aux clients de soumettre leur tâche dans une file d'attente FIFO, et de décider quel serveur doit effectuer la tâche. Un serveur ne peut effectuer qu'une tâche à la fois. Vous devez utiliser gRPC (gRPC Remote Procedure Call), un système libre d'appel de procédure à distance développé par Google pour implémenter votre gestionnaire.

Dans ce TP, vous vous contenterez d'implémenter un gestionnaire qui envoie un appel gRPC sur un seul serveur. Vous devrez utiliser le traçage système (avec LTTng) afin de visualiser l'exécution des appels gRPC. Pour cela, le code a été instrumenté.

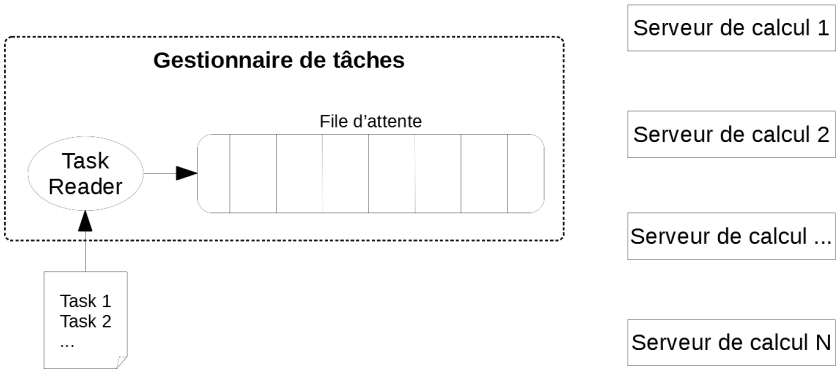

FIGURE 1 – Architecture

2.2 Mise en place

Avant de commencer vous devriez lire attentivement l'ensemble du TP afin de comprendre l'enchaînement des étapes, et vous informer de la section dépannage à la fin du TP.

2.3 Développement d'un programme RPC

Un code incomplet en C++ de l'application vous est fourni dans l'archive `TP2_trace_partie1.tar.gz` :

- `task_scheduler`
  - `lttng-traces`
    - `grpc_tracing.h` : déclaration des tracepoints
    - `grpc_tracing.c`
  - `manager.cc` : code du manager, compléter les **TODO**
  - `server.cc` : code du serveur, rien a modifier.
  - `operation.proto` : fichier d'IDL protocol buffers à compléter
  - `Makefile` : compile le serveur et le manager
  - `trace.sh` : lance la session lttng
  - `correction.sh` : script pour valider le TP

Vous devez le compléter la déclaration des interfaces IDL, et le code du manager, qui doit prendre en paramètre le nom de la tache (exemple d'exécution : `./manager tache45`). Vous pouvez éditer le code sur votre machine de labo ou personnelle, mais la compilation doit se faire dans le container déployé dans la VM (voir paragraphe suivant).

Après modification, refaire une archive `TP2_trace_partie1_modif.tar.gz`.

2.4 Déploiement de la machine virtuelle

La VM fournie (`VM_TP234_INF8480.ova`) est à déployer sur Virtualbox de la même manière que pour le TP1. Les ressources nécessaires sont bien plus faibles, vous pouvez normalement la déployer sur un PC avec 8 go de RAM. N'oubliez pas de supprimer la VM du TP1.

Mettez en place la la redirection de port invité 22 vers 2222 avec virtualbox.

Cette VM est accessible uniquement en SSH, avec le fichier de clé fourni avec la VM (il n'y a donc aucun mot de passe à rentrer). Vous devez utiliser la commande suivante depuis votre PC pour vous y connecter : (pas de mot de passe root, utilisez sudo)

```
[MONPC]$ ssh -i cle_vm_tp234  -p 2222 inf8480@localhost
```

2.5 Utilisation d'un container

Vous allez tester votre code sur un container docker, avec les bibliothèques C++ de GRPC et Protobuf déjà installées.

Dans la VM, la commande `sudo docker ps` vous permet de tester si le service Docker est bien démarré.

Pour accélérer le déploiement, une image docker `inf8480tp2:v1` est disponible sur la VM, et est basée sur l'image `https://hub.docker.com/r/grpc/cxx/dockerfile`.

La commande suivante vous permet de voir les images disponibles sur la VM :

```
inf8480@inf8480:~$  sudo docker images
```

Lancer maintenant un container nommé `inf8480tp2`, basé sur l'image `inf8480tp2:v1`.

```
inf8480@inf8480:~$ sudo docker run -dit --name inf8480tp2 inf8480tp2
    :v1
```

La commande `sudo docker ps` permet de vérifier que le container est bien "up and running".

2.6 Tests

Démarche de test :

— Téléverser l'ensemble des fichiers modifiés de l'application sur la machine virtuelle (commande à exécuter sur le PC hôte) :

```
[MONPC]$ scp -i cle_vm_tp234 -P2222 TP2_trace_partie1_modif.tar.
   gz inf8480@localhost:
```

Puis dans le container (commande à exécuter sur la VM) :

```
inf8480@inf8480:~$ sudo docker cp TP2_trace_partie1_modif.tar.gz
    inf8480tp2:/root
```

— Connectez vous ensuite à l'intérieur du container (commande à exécuter sur la VM) :

```
inf8480@inf8480:~$ sudo docker exec -it inf8480tp2 /bin/bash
```

Une fois connecté, vous obtenez un prompt du type :

```
root@9aff26152f31:/#
```

Déplacez vous dans le répertoire où les fichiers ont été transférés :

```
root@9aff26152f31:/# cd /root/
```

— Compilez votre trace provider dans le dossier `lttng-traces` :

```
gcc -I. -c grpc_tracing.c
```

— Reculer d'un dossier et compiler le programme avec `make` (le `Makefile` vous permet de compiler les deux programmes manager et server)

> **Remarque**
>
> Si votre code ne compile pas ou que celui-ci ne fonctionne pas lorsque vous le testez avec les étapes suivantes, vous pouvez vous servir de `scp` puis `docker cp` afin de transférer uniquement le fichier modifié (`operation.proto` par exemple) au lieu de re-transférer toute l'archive .

— Lancer le script `sudo sh trace.sh` (lance le traçage et le programme `server` en tâche de fond). Le script `trace.sh` permet de générer une trace système du fonctionnement de votre programme.
— Lancer votre serveur en arrière plan avec `./server &` et appuyez sur entrée pour avoir à nouveau le prompt.
— Envoyez plusieurs tâches au manager : (`./manager tache23`, `./manager tache45`). Vous devriez recevoir à chaque fois la réponse du serveur.
— Exécuter les commandes suivantes pour terminer le traçage :

```
lttng stop
lttng destroy
```

— Le fichier de trace générés en sortie est situé dans le dossier `trace_files`.
— Téléverser votre dossier de trace (le dossier contenu dans `trace_files`) sur votre ordinateur de labo (faire une archive, utilisez `docker cp` puis scp de la même manière que l'on a transféré les fichiers de l'hôte vers le container, mais cette fois du container vers l'hôte).
— Utilisez le logiciel `Trace Compass` (a télécharger ici : `https://www.eclipse.org/tracecompass/`) afin de visualiser la trace (voir annexe pour mode d'emploi) .
— vous devriez obtenir un résultat comparable :

<srch>	<srch>	<srch>	grpc	<srch>
12:43:29.569 731 511	ustchannel_0	0	lttng_ust_statedump:soinfo	baddr=0x7f0facec5000, sopath=/lib/x86_64-linux-gnu/librt-2.23.so,
12:43:29.569 835 334	ustchannel_0	0	lttng_ust_statedump:soinfo	baddr=0x7f0faccbd000, sopath=/usr/lib/x86_64-linux-gnu/liburcu-bp
12:43:29.569 939 305	ustchannel_0	0	lttng_ust_statedump:soinfo	baddr=0x7f0facab5000, sopath=/usr/lib/x86_64-linux-gnu/liburcu-cc
12:43:29.570 034 303	ustchannel_0	0	lttng_ust_statedump:soinfo	baddr=0x7f0fac7ac000, sopath=/lib/x86_64-linux-gnu/libm-2.23.so,
12:43:29.570 075 134	ustchannel_0	0	lttng_ust_statedump:end	context._vpid=1487, context._vtid=1492
12:43:29.612 914 570	ustchannel_0	0	grpc_tracing:manager_send	string=tache envoyee, context._vpid=1488, context._vtid=1488
12:43:29.628 382 324	ustchannel_0	0	grpc_tracing:server_start	id=50051, context._vpid=1487, context._vtid=1502
12:43:29.628 419 439	ustchannel_0	0	grpc_tracing:server_end	id=50051, context._vpid=1487, context._vtid=1502
12:43:29.628 778 815	ustchannel_0	0	grpc_tracing:manager_recv	string=tache envoyee, context._vpid=1488, context._vtid=1488

FIGURE 2 – Exemple de trace

— Si vous avez vu la trace de vos messages envoyés dans Trace Compass, vous pouvez terminer en exécutant le script de correction **dans le container** avec le code header moodle donné dans la première question du quiz sans les guillemets et sans le b
Exemple : `ubuntu@tp2:~/task_scheduler_correction$ ./correction.sh.x cDExMjAzOE1UazNOekEzTkUxVWF6Tk9lZw==`

```
./correction.sh.x code_header_moodle
```

Pensez à le rendre exécutable si besoin !

2.7 Vérification et remise

Vous venez de découvrir le traçage système, cette méthode qui permet d'instrumenter le noyau d'un système d'exploitation ou une application afin d'enregistrer une trace des évènements qui s'y déroulent. Le traçage permet d'analyser les performances d'une application ou d'un système distribué ou temps réel. On peut, en utilisant un logiciel d'analyse de traces comme Trace Compass

détecter les latences, le temps d'exécution d'une application... Le traçage est utilisé afin de détecter les problèmes de pertes de messages dans un système distribué par exemple (on détecte un message envoyé, son chemin vers la carte réseau) et en déduire quel est l'élément défectueux.

Un quiz moodle vous permet de déposer vos résultats.

3 Partie 2 : Analyse d'une trace système d'un système distribué

3.1 Mise en situation

Vous êtes analyste au service informatique de Poly, et on vous a signalé que le logiciel des ressources humaines était lent.

Le site est un système distribué, composé de trois parties (application cliente, serveur d'application et serveur d'authentification) qui sont des applications gRPC. Lors de la connexion, le client envoie une requête au serveur d'application, et ce dernier vérifie l'identité du client auprès du serveur d'authentification.

Vous devez donc trouver la cause et la durée des goulots d'étranglement. Les goulots d'étranglement peuvent être dans le client, le serveur, ou le serveur d'authentification.

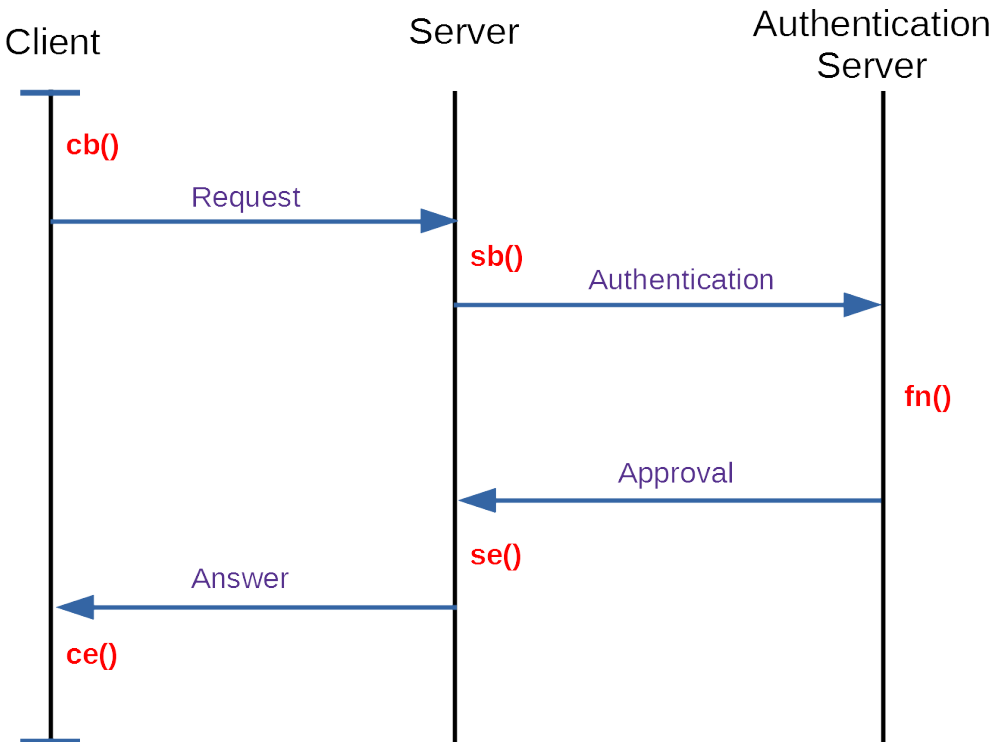

FIGURE 3 – Appel des fonctions

3.2 Analyse

Attention

Vous n'avez pas besoin de la VM dans cette partie ! vous devez utiliser Trace compass sur votre ordinateur personnel ou le PC du labo.

Vous devez analyser à l'aide de trace compass la trace fournie `1568729535905`. Vous devez calculer le temps passé dans le client, dans le serveur et dans le serveur d'authentification. Attention, le temps d'exécution réel du client n'est pas de `client_start` à `client_end` mais de `client_start` à `server_start` et de `server_end` à `client_end`. Même chose pour le temps d'exécution du serveur.

Vous pouvez voir le delta de temps entre deux évènements avec trace compass, en sélectionnant deux évènements, avec les touches `Ctrl + Maj` enfoncées.

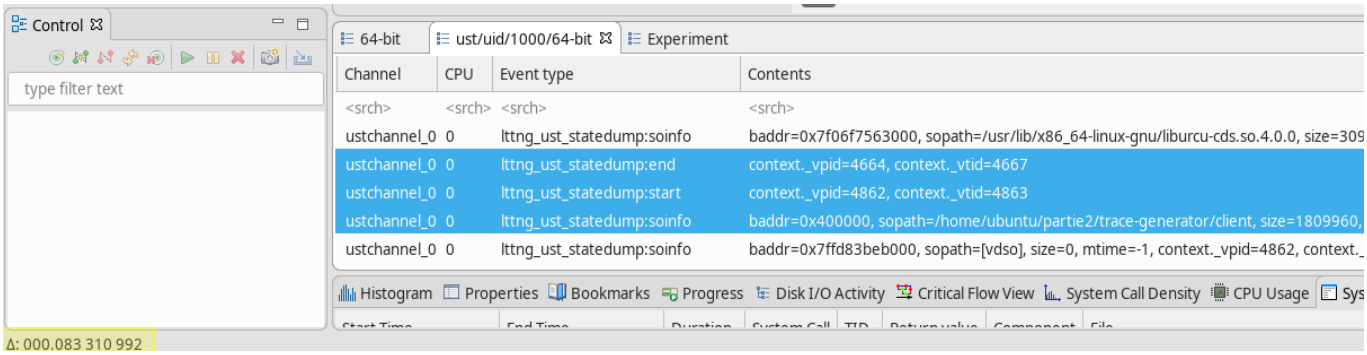

FIGURE 4 – Voir le delta de temps entre deux évènements (en bas à gauche)

3.3 Vérification et remise

Vous devez analyser la trace fournie, et répondre aux 3 questions questions du quiz Moodle proposées sur le site du cours.

4 Annexes

4.1 Traçage système

4.1.1 Définition

Une trace d'un programme est une représentation de l'exécution de ce même programme. Le but est d'instrumenter et d'optimiser la qualité des programmes en termes de performances et de robustesse.

4.1.2 LTTng

LTTng est un outil de traçage et de visualisation des évènements produits à la fois par le noyau linux et par les applications.

— **Définition :** `https://lttng.org/docs/v2.10/#doc-what-is-tracing`
— **Tracepoint provider :** `https://lttng.org/docs/v2.10/#doc-tracepoint-provider`

4.1.3 Trace Compass

Trace compass est un outil de visualisation graphique de traces systèmes.

Pour ouvrir une trace, cliquez sur *File* → *Trace Import*, cliquer sur *Browse...* et sélectionner le dossier de la trace.

Puis cocher la case du dossier de la trace à ouvrir, et enfin cliquer sur finish. Le menu à gauche, permet de naviguer entre les traces systèmes et traces applications (ust). Dans ce tp, vous analyserez les traces ust (userspace).

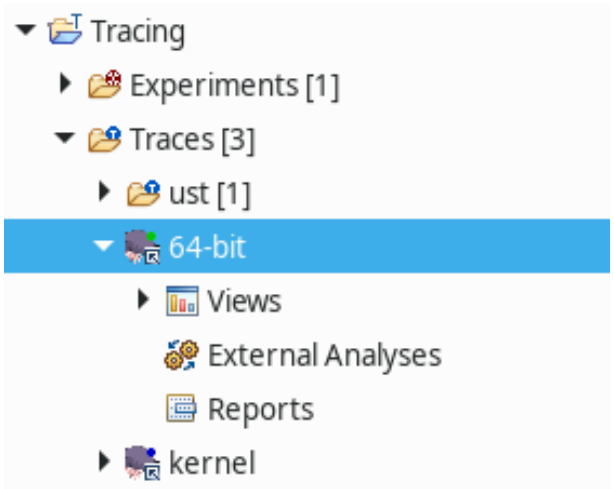

FIGURE 5 – Ouvrir la trace userspace

Vous pouvez le télécharger ici : `https://www.eclipse.org/tracecompass/`

♢