Projet – Calculateur réparti

1 Introduction

On désire réaliser un système de calcul distribué, c'est-à-dire un système permettant de déterminer des valeurs $f_k(x_1, x_2 \dots x_n)$, le calcul des f_k étant effectué sur des « nœuds » spécialisés. Le système doit permettre l'ajout ou le retrait de nœuds, voire la panne d'un nœud sans provoquer de perturbation s'il y a une redondance.

2 Description

Le système à réaliser repose sur deux programmes distincts : l'orchestrateur et les nœuds de calcul.

2.1 Nœuds de calcul

Un nœud de calcul est spécialisé pour une fonction f_k donnée. Son rôle est double :

— annoncer périodiquement (par exemple toutes les 10 secondes) à l'orchestrateur son existence et la fonction f_k calculée ainsi que le nombre de paramètres requis ;
— recevoir une requête de calcul de l'orchestrateur (avec les arguments $x_1, x_2 \dots x_n$), effectuer le calcul demandé et renvoyer le résultat à l'orchestrateur.

Plusieurs nœuds peuvent annoncer la même fonction f_k, ce qui permet d'obtenir de la redondance. Vous aurez autant de programmes que de fonctions f_k, même s'ils partagent la majeure partie de leur code.

2.2 Orchestrateur

L'orchestrateur est le programme central : il est le point de convergence de plusieurs flux :

— les expressions saisies par l'utilisateur ;
— les annonces des nœuds de calcul ;
— les retours des calculs des nœuds, c'est-à-dire les valeurs $f_k(x_1, x_2 \dots x_n)$ calculées par les nœuds.

Dès que l'orchestrateur commence son exécution, et pendant toute son existence, il collecte les annonces des différents nœuds de calcul et maintient la liste des nœuds et les fonctions f_k correspondantes. S'il constate qu'un nœud n'a pas envoyé d'annonce pendant une durée d_a (par exemple, $d_a = 35$ secondes), il suppose que le nœud n'est plus disponible et le retire de sa liste.

Lorsque l'orchestrateur reçoit une ligne de l'utilisateur (sur l'entrée standard), il consulte sa liste des nœuds pour voir s'il y en a un qui propose le calcul de cette fonction et n'est pas déjà occupé par un calcul. Si c'est le cas, il lui envoie les arguments et se remet en attente. Dans le cas contraire, le calcul est simplement refusé. Lorsque l'orchestrateur reçoit le résultat d'un calcul, il l'associe à la demande originale de l'utilisateur et affiche la demande et son résultat.

Optionnel : si l'orchestrateur a envoyé un calcul à un nœud qui est devenu hors service au bout du délai d_a ci-dessus, l'orchestrateur doit alors soit trouver un autre nœud pour le calcul, soit afficher un message d'erreur si aucun nœud n'est disponible pour la fonction f_k demandée.

3 Implémentation

On implémentera des nœuds à minima pour les 4 opérations élémentaires (f_1 = addition, f_2 = soustraction, f_3 = multiplication et f_4 = division) sur des entiers. Comme ces opérations sont rapides, on introduira un délai pseudo-aléatoire compris entre 0 et d_r (par exemple $d_r = 50$ secondes).

Le système décrit ci-dessus a vocation à être déployé sur un réseau local et non sur un réseau longue distance ; de plus, les messages échangés sont très courts. Enfin, une extension possible (voir ci-dessus) concerne le *broadcast*. On utilisera donc obligatoirement le protocole UDP.

Pour faciliter le développement, même avec un seul ordinateur sur lequel tournent les différents programmes, on utilisera un dispositif simplifié dans lequel l'orchestrateur utilise un port UDP fixe et chaque nœud connaît l'adresse IP et le port de l'orchestrateur. De plus, comme les nœuds peuvent utiliser la même adresse IP, l'orchestrateur utilisera un couple <adresse IP, port> pour repérer chaque nœud.

Optionnel : dans le cas où on utilise plusieurs ordinateurs, vous pouvez prévoir un dispositif d'annonce par *broadcast*. Dans ce cas, les nœuds n'ont pas besoin d'avoir connaissance de l'adresse IP de l'orchestrateur.

4 Exemple

Voici un exemple d'utilisation de l'orchestrateur :

```
turing> ./orchestrateur
...
orchestrateur> +(2,3)                    # demande effectuée à 14:33:20
orchestrateur> *(6,5)                    # demande effectuée à 14:33:25
orchestrateur>
14:33:25 *(6,5) = 30 by <2001:660:4701:2001::100,1233> at 14:33:32
orchestrateur>
14:33:20 +(2,3) = 5 by <2001:660:4701:2001::100,1232> at 14:34:03
orchestrateur> /(10,3)
Aucun nœud disponible pour /
```

5 Travail demandé

Le projet est à réaliser en langage C ou C++ à l'aide des *sockets* (vous n'utiliserez pas d'autre mécanisme de programmation réseau). Bien évidemment, vos programmes doivent être compatibles IPv4 et IPv6. Vos programmes doivent pouvoir fonctionner sur une machine Linux comme `turing`[1].

Pour simplifier, on ne tiendra pas compte des problèmes de sécurité et d'authentification.

Il vous est demandé :

- de rédiger une description du protocole (différents messages échangés, format) que vous implémentez entre les acteurs, en justifiant vos choix. Votre description doit être suffisamment précise pour permettre à quelqu'un d'autre que vous de programmer un composant quelconque sans faire référence à vos fichiers sources ;
- de programmer les composants de l'application en langage C ou C++ sous Unix, avec les sockets.

6 Modalités de remise

Ce projet est à réaliser par groupe de deux. Il vous sera demandé de démontrer votre contribution personnelle au projet. Toute copie entre groupes ou inspiration extérieure sera sanctionnée.

La date de remise est le 12 Novembre.

Vous déposerez sur Moodle, dans l'espace de devoirs prévu à cet effet, votre projet (documentation, sources, Makefile) sous forme d'une archive au format « *login*.tar.gz » où *login* est votre nom de login sur `turing`. Vous prendrez soin de supprimer tous les fichiers binaires (exécutables, objets) autres que votre rapport en format PDF.

7 Soutenance

Vous devrez faire une démonstration, qui sera suivie d'une discussion de quelques minutes. Lors de la discussion vous aborderez les points suivants :

- les principes généraux de fonctionnement de votre système et les éléments clefs du protocole que vous avez conçu ;
- les éléments notables sur l'implémentation (vous n'oublierez pas de mentionner les limites de votre implémentation, ou à l'inverse les points supplémentaires que vous auriez éventuellement ajoutés) ;

1. On notera que le nom DNS `turing6.u-strasbg.fr` référence spécifiquement l'adresse IPv6 de la machine `turing`.

— le bilan du projet (éléments acquis, difficultés rencontrées, etc.).