Messages en parallèle

Luis D. Pedrosa & Jean-Cédric Chappelier 2017

Jusqu'ici les clients envoyaient leurs messages en série, un après l'autre. Cela permet de simplifier le code mais empêche les serveurs de travailler de façon concurrente, ce qui limite la performance globale du tout. Nous allons cette semaine changer cela et permettre aux clients de lancer tous leurs messages à la fois et de récupérer et traiter les réponses quand elles arrivent.

Concrètement, l'opération « put » va maintenant envoyer ses N messages d'un coup (sans se préoccuper des réponses) **puis** déclarera un succès dès que W réponses (positives) auront été reçues (dans le temps imparti lié à l'ouverture du socket correspondant ; sinon, comme avant, c'est un échec ; ce point là ne change pas).

De même, l'opération « get » va aussi envoyer ses N messages d'un coup (sans se préoccuper des réponses) **puis** déclarera un succès dès que R réponses identiques auront été reçues (dans les mêmes condition de délai que précédemment).

Mais du coup, il se peut tout à fait qu'un serveur (typiquement un peu lent) continue de répondre à une requête déjà terminée, interférant du coup peut être avec une *autre* requête du même client. Pour éviter toute confusion, il va maintenant falloir (si ce n'est pas déjà fait comme cela), ouvrir un *nouveau* socket UDP pour chaque opération « put » ou « get » effectuée (mais **pas** pour chaque serveur contacté pour une même opération !). Cela évitera au messages UDP d'une ancienne opération d'être reçu par une nouvelle.

Concrètement, il va vous falloir réviser :

- (`client.h` et) `client.c` : le client n'a maintenant plus un seul socket associé ; on peut donc le supprimer (pas nécessaire, mais on peut) de la structure `client_t`, mais surtout ce n'est plus au client (`client_init()`) d'ouvrir ce socket ;
- `network.c` : c'est ici qu'il y a le plus « gros » du travail (ce n'est pas très compliqué non plus) : puisque c'est maintenant à `network_get()` et `network_put()` d'ouvrir un nouveau socket à chacun de leur appel, puis de changer la politique de communication comme expliqué ci-dessus ;
- `pps-list-nodes.c` : il faut en effet appliquer ici la même logique générale : envoyer toutes les requêtes une après l'autre, **puis** ensuite collecter les réponses ; comme maintenant elles peuvent arriver dans n'importe quel ordre, il faut aussi les remettre dans le bon ordre (le même que celui de la

semaine passée : par adresse/port croissant).

IV. Rendu

Le travail de cette semaine termine complètement l'aspect code de votre projet proprement dit et correspond au code à rendre en fin semestre (dernier rendu, délai : le lundi 04 juin 23:59). La semaine prochaine en effet il n'y aura plus de code à produire sur le projet lui-même, mais uniquement pour une petite étude exploitant et valorisant votre travail. Ce rendu final (code de cette semaine + étude de la semaine prochaine) compte pour 40 % de la note finale (comme indiqué dans le barème).

Pour rendre cette partie du code (et le code de l'étude de la semaine prochaine), mettez (et commit+push) tous votre code correspondant dans le répertoire `done/` de votre dépôt GitHub (de groupe et à la racine, **PAS** dans `provided/` ; vous ne devez **jamais** modifier le répertoire `provided/`) puis « tagger » le avec le tag `projetFinal` (avec **une seule** majuscule au milieu, pas de underscode (« souligné ») ; **AUCUN** autre tag ne sera considéré comme rendu du projet) et « pousser » le résultat vers GitHub (`git push --tags`).

Le délai pour faire le `push` de rendu est : le **lundi 04 juin 23:59**.

ATTENTION ! Pour pouvoir être accepté, votre rendu devra :

1. être proprement tagé (tag `projetFinal`) ;
2. se trouver dans le répertoire `done/` ;
3. compiler avec `make`.

Comme d'habitude, si vous souhaitez obtenir du feedback, taggez votre projet avec le tag `week12`.

Pour la participation à l'étude de la semaine prochaine, nous prendrons simplement l'état de votre `master` chaque jour entre le 30 mai et le 4 juin (détails la semaine prochaine).

Vous pouvez donc commencer par tagger votre version « finale » par le tag `week12` (pour le feedback) ou attendre de voir comment votre release `master` se comporte la semaine prochaine pendant l'étude et y ajouter d'éventuelles corrections avant de faire le tout dernier tag de rendu (tag `projetFinal`).