Anneau des clés

Luis D. Pedrosa & Jean-Cédric Chappelier 2017

I. Description générale

Cette semaine constitue l'aboutissement de la méthode générale des table de hashage distribuée (DHT) : l'anneau des clés (revoir si nécessaire le fichier de description principal) : plutôt que de sélectionner les *nœuds* (qui vont donc maintenant être distincts de la notion de serveur) suivant l'ordre dans lequel les serveurs apparaissent dans le fichier `servers.txt`, nous allons les positionner dans l'anneau des clés (un serveur pouvant alors maintenant avoir *plusieurs* positions dans cet anneau).

Ce changement implique que le format du fichier `servers.txt` lui-même doit être modifié : il y aura toujours un seul serveur par ligne, mais nous ajoutons maintenant une information supplémentaire : le nombre de nœuds qu'un serveur va offrir. Le format d'une ligne devient donc :

```
"<IP> <Port> <# of Nodes>\n"
```

par exemple :

```
127.0.0.1 1234 3
```

Comme expliqué dans le fichier de description principal, tous les clients doivent maintenant déterminer *les* position_s_ des serveurs (= les nœuds) non plus à partir de leur ligne dans le fichier `servers.txt`, mais en calculant une valeur SHA-1. De même, le positionnement d'une clé se fera en calculant son SHA-1. Les deux valeurs de SHA-1 obtenue seront directement interprétées comme des positions (nombres entiers) et directement comparée.

Pour calculer le SHA-1 d'un nœud d'un serveur, on calculera le SHA-1 de la chaîne

```
"<IP> <Port> <Node-ID>\n"
```

(avec espaces), où `<Node-ID>` va de 1 au nombre de nœuds indiqués dans le fichier `servers.txt` pour le serveur en question.

Par ailleurs, lors de la sélection de (au plus) N nœuds dans les algorithmes des clients, faites maintenant bien attention de sélectionner des nœuds de serveurs différents (sautez les nœuds de serveurs déjà visités ; même IP **et** même port).

II. Implémentation

Toujours dans l'esprit de bien modulariser le code, nous proposons donc d'implémenter les fonctions nécessaires à la gestion de l'anneau des clés dans `ring.c`. Commencez par parcourir `ring.h` et lire les commentaires. Le cœur de ce travail est bien sûr la fonction `ring_get_nodes_for_key()` ayant pour but de retourner la listes des N nœuds (N passé en paramètre) répondant à une clé donnée (et vérifiant la contrainte évoquée ci-dessus : tous des nœuds de serveurs *différents*). L'implémentation de cette fonction pourra elle-même être modularisée suivant vos choix.

`ring_init()` devra, bien sûr, lire le fichier `servers.txt` et construire la représentation de l'anneau des clés (liste de nœuds) en conséquence. Pensez à utiliser `node_list_sort()` et `node_cmp_sha()` pour les mettre dans l'ordre souhaité pour l'utilisation qu'on en fait ici (c'est le moment d'écrire ces deux fonctions si vous ne l'avez pas encore fait ; `node_list_sort()` est dans `node_list.[ch]` et `node_cmp_sha()` dans `node.[ch]`. Pensez à utiliser la fonction standard `qsort()` ici. Il faudra aussi modifier la structure `node_t` dans `node.h` et la fonction `node_init()` dans `node.c` pour prendre en compte les SHA-1 ; revoir si nécessaire ce que vous avez fait en semaine 3).

[**Note :** ne vous préoccupez pas du « `_unused` » sur le paramètre `node_id` de `node_init()` dans `node.h` : ce paramètre doit maintenant effectivement être utilisé (pour calculer le SHA-1) ; « `_unused` » ne veut pas dire que le paramètre n'est pas utilisé, mais qu'il *pourrait* ne pas être utilisé (ce qui était le cas jusqu'à maintenant) et que donc le compilateur ne doit pas mettre d'avertissement s'il n'est pas utilisé ; mais ce paramètre peut tout à fait être utilisé ; donc : ne vous préoccupez pas de ce « `_unused` » **]

Une fois `ring.c` terminé (ou fait par votre binôme), modifiez `client.h`, `client.c` et `network.c` pour utiliser des `ring` au lieu des serveurs et autres `node_list` utilisés jusqu'à présent. (ces modifications devraient être simples; tout le travail a été fait dans `ring.c`.)

Modifiez enfin la commande `pps-list-nodes()` pour :

- qu'elle liste les noeuds par adresses de serveurs (voir exemple ci-dessous) ; nous vous conseillons pour cela d'écrire et utiliser la fonction `node_cmp_server_addr()` proposée dans `node.h` ;
- qu'elle affiche le SHA-1 du noeud (voir exemple ci-dessous).

Enfin, je vous rappelle que pour calculer les SHA-1, il faut compiler avec la bibliothèque `-lcrypto` (cf semaine 3).

III. Exemple

Par exemple, si le fichier `servers.txt` contient :

```
127.0.0.1 1234 3
127.0.0.1 1235 3
127.0.0.1 1236 3
```

et que l'on lance les trois serveurs :

```
echo "127.0.0.1 1234" | ./pps-launch-server &
echo "127.0.0.1 1235" | ./pps-launch-server &
echo "127.0.0.1 1236" | ./pps-launch-server &
```

alors la commande `./pps-list-nodes` donnera :

```
127.0.0.1 1234 (a902e3a5aa4f73150f459436b0580cb7ad72b566) OK
127.0.0.1 1234 (aa66f3e5a8d9cdc5c0bd49708bc59847e6915634) OK
127.0.0.1 1234 (e9b9c7e5d1569abaf1dc5b0ce2958f14ef831770) OK
127.0.0.1 1235 (1bcd2db55b43d8c6b50583892f141a6bb3224c04) OK
127.0.0.1 1235 (914f6ade5b49a3a9be257f8a56bbde9a83fa46aa) OK
127.0.0.1 1235 (c484ea9b3b14d139b1456032a49990367b857fe6) OK
127.0.0.1 1236 (5f26268754fcf2a51fcfacaaa2aaf4f0d83f6d67) OK
127.0.0.1 1236 (93149f866bf3acc9710375cb46706bf09960a6ab) OK
127.0.0.1 1236 (ee482fd6bcd8a1eb2a929a0d284b563404b64d19) OK
```

L'anneau des clé est donc dans ce cas (ordre des nœuds) :

```
1bcd2db55b43d8c6b50583892f141a6bb3224c04 (127.0.0.1 1235)
5f26268754fcf2a51fcfacaaa2aaf4f0d83f6d67 (127.0.0.1 1236)
914f6ade5b49a3a9be257f8a56bbde9a83fa46aa (127.0.0.1 1235)
93149f866bf3acc9710375cb46706bf09960a6ab (127.0.0.1 1236)
a902e3a5aa4f73150f459436b0580cb7ad72b566 (127.0.0.1 1234)
aa66f3e5a8d9cdc5c0bd49708bc59847e6915634 (127.0.0.1 1234)
c484ea9b3b14d139b1456032a49990367b857fe6 (127.0.0.1 1235)
e9b9c7e5d1569abaf1dc5b0ce2958f14ef831770 (127.0.0.1 1234)
ee482fd6bcd8a1eb2a929a0d284b563404b64d19 (127.0.0.1 1236)
```

Et donc si vous faites par exemple des `debug_print` de `ring_get_nodes_for_key()` pour la clé « coucou » (dont le SHA-1 est « `5ed25af7b1ed23fb00122e13d7f74c4d8262acd8` », cf la semaine 03) vous devriez obtenir les nœuds suivants (dans cet ordre ; le nombre dépendant bien sûr du N que vous utilisez) :

```
127.0.0.1 1236 (5f26268754fcf2a51fcfacaaa2aaf4f0d83f6d67)
127.0.0.1 1235 (914f6ade5b49a3a9be257f8a56bbde9a83fa46aa)
127.0.0.1 1234 (a902e3a5aa4f73150f459436b0580cb7ad72b566)
```

IV. Conseil et rendu

Le travail de cette semaine ne sera pas évalué séparément, mais devra faire partie du dernier rendu du projet (délai : le lundi 04 juin 23:59).

Par ailleurs, n'oubliez pas ce dimanche 13 mai de faire le cinquième rendu du semestre (second rendu sur le projet) correspondant au travail effectué jusqu'à la semaine passée (semaine 10), à rendre sous le tag `projet02`.