Benchmarking

Luis D. Pedrosa, J.-C. Chappelier, 2017

Buts du travail de cette semaine

Si tout a été comme prévu, vous devriez maintenant avoir une table de hashage distribuée (DHT) totalement opérationnelle. Le but du travail de cette semaine est de vous permettre de mettre en valeur le résultat de vos efforts passés, dans un cadre expérimental plus réaliste. Le sujet de cette semaine est volontairement plus ouvert ; libre à vous de l'adapter, l'approfondir, etc. ; mais **attention au temps passé !** Cela doit rester raisonnable par rapport au barème (10%, voir en fin de cette page) et aux 2 crédits de ce cours !

Le but du travail de cette semaine est donc de déployer votre travail dans un environnement large à beaucoup de serveurs (voir ci-dessous) et d'expérimenter en détails ses performances ; en particulier comment elles sont affectées par les paramètres de redondance N, W et R , ainsi que d'autres facteurs tels que la longueurs des clés, des valeurs, la taille des tables locales, etc.

Alors que jusqu'à maintenant vous n'avez déployé et testé votre DHT que localement, nous allons cette semaine déployer tous les serveurs de tous les groupes sur diverses machines (de l'Ecole). Nous nous occupons de déployer vos serveurs et vous fournirons, via votre dépôt GitHub (on le remettra éventuellement à jour si nécessaire ; voir plus bas), le fichier `servers.txt` à utiliser pour accéder à cette grande DHT. Le projet de chaque groupe dans l'état du `master` aux moments où nous lancerons le tout sera déployé dans cette grande DHT et votre client de scénarios de tests (ci-dessous) devra donc interagir potentiellement avec les serveurs de plusieurs autres groupes.

En utilisant cette infrastructure, vous allez effectuer différentes expériences et mesures de performances de différents scénarios d'utilisation. Pour cela, vous allez devoir mesurer combien de temps s'écoule entre différentes portions de votre code de test/d'expérience (client ; il n'y a *rien* à faire sur les serveurs). Vous pouvez faire de telles mesures en utilisant la fonction `clock_gettime()` avec la « clock » `CLOCK_MONOTONIC` et soustraire les deux mesures que vous obtenez à chaque extrémité de la zone de code dont vous souhaiter mesurer la durée d'exécution. Cette différence est exprimée en nanosecondes écoulées. En écrivant cette valeur dans un fichier, vous pouvez ainsi collecter différents échantillons de mesure pour vos expérience. Ce fichier de durées vous servira ensuite, à l'aide

d'outils externes détaillés plus bas, à produire des courbes globales de synthèse de performances.

Exemple d'utilisation de `clock_gettime()` :

```
static void print_time(const struct timespec* p_time_start,
                       const struct timespec* p_time_end)
{
    long nsec = p_time_end->tv_nsec - p_time_start->tv_nsec;
    while (nsec < 0) nsec += 1000000000;
    printf("%ld%09ld\n", p_time_end->tv_sec - p_time_start->tv_sec, nsec);
}


...
    struct timespec time_start, time_end;
    int time_start_ret = clock_gettime(CLOCK_MONOTONIC, &time_start);
    M_EXIT_IF_ERR(time_start_ret, somename);

    error_code ret = some_operation(key, value);
    M_EXIT_IF_ERR(ret, somename);

    int time_end_ret = clock_gettime(CLOCK_MONOTONIC, &time_end);
    M_EXIT_IF_ERR(time_end_ret, somename);

    print_time(&time_start, &time_end);
...
```

Note : `CLOCK_MONOTONIC` n'est pas du C standard, mais POSIX. Si vous voulez compiler sans erreur avec une option `-std=` du compilateur, ajoutez simplement la ligne suivante au début de votre `.c` (*avant* les `#include`) :

```
#define _POSIX_C_SOURCE 199309L
```

[**fin de note**]

Etape 1 : concevoir et écrire des scénarios d'expérience

La première étape consiste donc à écrire du code C correspondant à différents scénarios que vous souhaitez tester, typiquement des séquences de `put` et `get` vers la DHT. Vous pouvez faire varier différents aspects du système pour étudier la dépendance des performances par rapport à ces paramètres. Par exemple, vous pouvez changer N, W et R ; ou encore changer le scénario lui-même (plus ou moins de `put` ou `get`, clé ou valeurs plus ou moins longues, variées/répétées, etc.).

Il s'agit donc simplement d'écrire un code similaire au cœur de `pps-client-get.c` et `pps-client-put.c` (pas *nécessaire* de faire de la gestion d'arguments ici ; c'est comme vous voulez), d'y mettre des `clock_gettime()` bien placés et de reporter leur valeur (dans un fichier ou sur stdout en utilisant la redirection du Shell : `./macommande > un_fichier`).

Etape 2 : produire des courbes de synthèse

Le résultat final attendu pour cette étape est un court rapport (**3 pages maximum**) expliquant les expériences que vous avez conduites et surtout des courbes illustratrices des effets que vous avez pu observer. Dans ce type d'expériences/de rapports, il est fondamental de non seulement reporter les valeurs moyennes, mais aussi la variabilité de celles-ci (indication de la variance, de barres d'erreurs, etc.). Une mesure de moyenne **sans** indication de sa variabilité est totalement nulle et inexploitable (même si c'est malheureusement tellement souvent le cas dans la presse...).

Pour réaliser de tels graphes, vous pouvez utiliser l'un ou l'autre des outils suivants : octave/matlab, gnuplot, LibreOffice/Excel, R, plotly/matplotlib, ...

Exemple de scénarios

Le travail de cette semaine est volontairement très ouvert et constitue un exercice de « pensée critique/analytique » telle que, en tant qu'ingénieur, vous aurez à produire dans votre métier. Nous vous recommandons donc à réfléchir par vous-même (dans votre groupe) à des scénarios de test intéressants. Le but n'est pas ici la quantité (ne vous noyez pas sous le nombre de scénarios et sous les lignes de code) mais la *qualité* (montrer un ou deux effets clairs).

Pour vous guider dans votre réflexion, nous vous proposons quelques bases d'expérimentation possibles :

1. dans un scénario équilibré (autant de `put` que de `get`), prendre $R{=}W{=}N/2{+}1$ et mesurer le temps de latence pour différentes valeurs de N, par exemple de 1 à 10 ;
2. dans un scénario de `put` uniquement, prendre $N{=}10$ et mesurer le temps de latence pour différentes valeurs de W, par exemple de 1 à 10 ;
3. dans un scénario de `get` uniquement (après, bien sûr, une phase initiale non mesurée de `put` !), prendre $N{=}10$ et mesurer le temps de latence pour différentes valeurs de R, par exemple de 1 à 10 ;
4. avec $N{=}3$, $W{=}R{=}2$, mesurer le temps de latence pour différents scénarios `put`/`get` :
    1. lancer un premier round de `put` non mesurés pour initialiser le système ;

    2. lancer 100 opérations `put` ou `get`, dans un ordre aléatoire avec des ratios différents :
        - 0 `put` et 100 `get`
        - 10 `put` et 90 `get`
        - etc.
        - 100 `put` et 0 `get`
    3. répéter 10 fois le groupe des onze étapes précédentes avec différents tirages aléatoires pour avoir des estimations statistiques (moyennes et variances/barres d'erreur) ;
5. avec N=3, W=R=2 et un scénario équilibré, faites varier la taille des clés (à taille de valeurs fixe) ;
6. avec N=3, W=R=2 et un scénario équilibré, faites varier la taille des valeurs (à taille de clés fixe).

Déroulement des expériences

1. La DHT de tous vos serveurs tournera
   du jeudi 24 mai midi
   au lundi 4 juin 23h00 ;
2. Elle sera(/pourra être) arrêtée tout les jours entre 20h00 et 20h30
   Il ne faudra donc **PAS** faire tourner d'expérience pendant cette demi-heure ;
3. Nous récupèrerons l'état de votre master
   le jeudi 24 mai dans la matinée
   puis chaque jour à 20h00,
   et (re)compilerons votre server (`pps-launch-server`) et lancerons 3 instances (3 servers différents donc pour chaque groupe), chacune servant 3 nœuds ;
4. Nous ferons un reset complet (rédemarrage du tout) chaque jour à 20h30 (entre 20h00 et 20h30 pour être précis).
5. Hors de la période 20h00-20h30, vous aurez donc tout loisir de votre coté de lancer autant de fois que vous le voulez votre/vos client(s) de scénarios d'expériences.

Avant le début de ces expériences (jeudi 24 mai midi), nous vous encourageons bien sûr à tester vos scénarios localement sur votre propre machine.

Rendu

Le rendu de cette dernière étape du projet sera un **court** rapport mettant en évidence votre analyse des résultats obtenus (**3 pages maximum**, format PDF, dans une fonte raisonnablement lisible ; à placer dans `done/rapport.pdf`). Nous attendons ici non seulement des résultats (intéressants) eux-mêmes, mais aussi

votre commentaire critique et une mise en perspective. Discutez la vraisemblance des résultats obtenus, les tendances, les anomalies et les limites de votre approche.

Concluez en soulignant les limitations pratiques que vos expérience auront pu mettre en exergue et proposant des suggestions sur comment de telles limitations pourraient en pratique être résolues ou réduites par des compromis.

L'évaluation de ce rapport comptera pour 10% de la note (soit un quart des 40% du rendu final tel qu'annoncé dans le barème).

Ce rapport ainsi que le(s) code(s) du/des scénario(s) devront être tagés, avec tout le reste du code jusqu'à la semaine passée, avec le tag `projetFinal`. Le délai pour ce rendu final est **le lundi 04 juin 23:59**.