NF26 — Stockage en haute volumétrie et applications

TD1 — Stockage clef-valeur, *sharding* et *IO-bound*

P2021 — 18 mars 2021

Florent Chehab, Julien Jerphanion

Table des matières

1 Prise en main de l'environnement

1.1 Présentation

Pour les TDs de NF26, nous mettons à votre disposition un environnement d'exécution comprenant 3 serveurs :

— `nf26-1.leger.tf`
— `nf26-2.leger.tf`
— `nf26-3.leger.tf`

Sur chacune de ces machines, vous avez un utilisateur nominatif auquel vous pouvez vous connecter grâce aux identifiants qui vous ont été envoyés :

```
ssh <login_cas>@nf26-{1,2,3}.leger.tf
```

Vous êtes invités à vous répartir aléatoirement sur ces serveurs.

Nous nous assurons que les TDs sont réalisables sur ces serveurs, et ils seront essentiels par la suite pour notre usage de `Spark`, `Cassandra`, etc.

1.2 Synchronisation de votre code

Il est fort probable que vous souhaitiez développer sur votre machine en local en utilisant votre éditeur de code préféré; tout en synchronisant votre code et en l'exécutant depuis un des serveurs.

Si quand nous disons cela, vous vous sentez perdus, nous pouvons vous recommandez :

- `VSCode` (https://code.visualstudio.com/) et "Remote SSH" (voir https://code.visualstudio.com/docs/remote/ssh pour la configuration)
- `PyCharm Edu` (https://www.jetbrains.com/fr-fr/pycharm-edu/) et "SSH Interpreter" (https://www.jetbrains.com/help/pycharm/configuring-remote-interpreters-via-ssh.html#ssh)
- `rsync` (https://fr.wikipedia.org/wiki/Rsync)

N'hésitez pas à vous entraider si vous avez des problèmes de configuration. Vos chargés de TDs seront également disponibles.

1.3 Support et demande d'aide

Pour vous aider lorsque vous rencontrez des bugs ou problèmes sur le serveur pour les TDs, nous passerons par :

- (Recommandé) Utilisation de `tmux` (https://doc.ubuntu-fr.org/tmux) directement sur le serveur, afin de pouvoir travailler dans une même session `tmux`). **Prenez soin d'en maitriser les bases (split de terminaux et déplacement d'un terminal à un autre).**
- (Optionnellement) `VSCode Live Share` pour les utilisateurs et les utilisatrices de `VSCode` (https://marketplace.visualstudio.com/items?itemName=MS-vsliveshare.vsliveshare-pack)

1.4 *Git* et *Gitlab*

Nous aimons `Git`. L'ensemble des rendus se dérouleront via des projets nominatifs sur l'instance `Gitlab` de l'UTC.

Même si vous êtes seuls, vous devrez utiliser le système de branche et passer par des pulls requests vers la branche `main` depuis `Gitlab`.

Pour ce premier TD, votre projet `Gitlab` personnel est https://gitlab.utc.fr/NF26/tf-managed/TDs-p2021/td-key-value-part1/td-key-value-part1-<login_cas>.

2 Implémentation de solutions de stockage

On peut donc maintenant attaquer les aspects pratiques de ce TD !

Afin de mettre en pratique le cours sur le stockage de données clef / valeur, nous allons implémenter deux variantes de ce système de stockage.

2.1 Principe

Une fois le code du projet associé à ce TD cloné depuis votre projet nominatif, vous pouvez utiliser les commandes :

```
make help
make init
make style
make test
make run-benchmark
```

Pour toute cette partie 2, nous vous demandons de compléter le code dans le fichier `/key_value_part1/simple_crud_storage.py` afin d'implémenter les opérations *CRUD* de telle sorte à ce que les tests fonctionnent.

Vous pouvez lancer ces tests avec :

```
make test
```

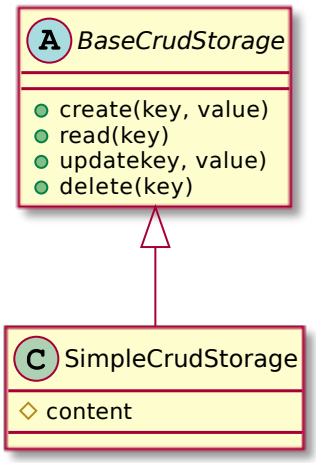
Vous êtes libre de rajouter des tests. **Soyez vigilants au fait que les tests utilisés par le système automatisé d'évaluation seront différents et plus exhaustifs.** Pour connaitre les critères d'évaluation, rendez-vous en section 4.2.

Initialement, 1 seul test doit passer.

2.2 Solution de stockage clef/valeur CRUD

Nous avons défini une classe abstraite `BaseCrudStorage` qui définit une interface attendue pour toutes les classes de stockage implémentant les opérations *CRUD*.

Dans un premier temps il s'agit d'implémenter la classe de stockage *CRUD* simple, nommée `SimpleCrudStorage`, basé sur l'utilisation en python d'un objet interne de type `dict`.

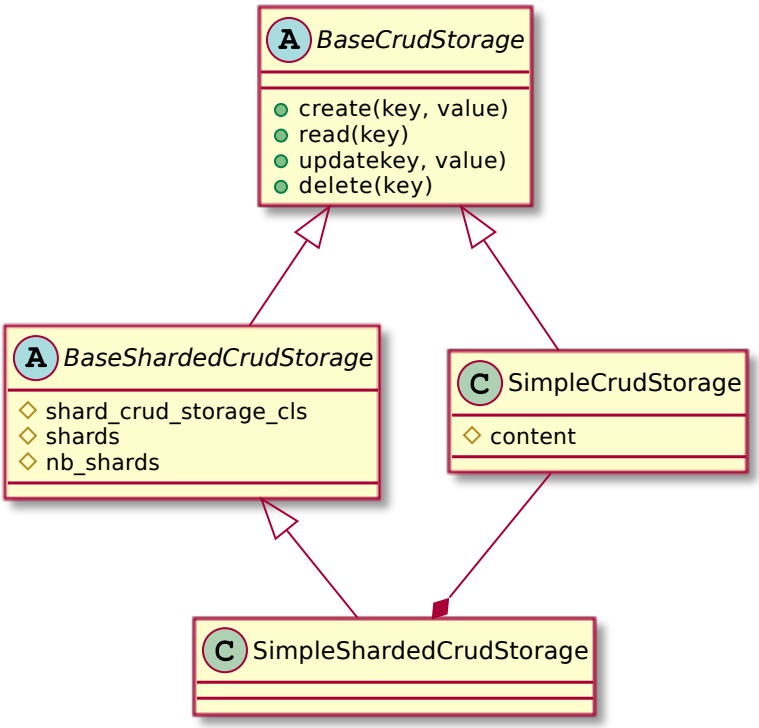


Une fois cela fait, 17 tests doivent passer.

2.3 Solution de stockage clef/valeur CRUD et *sharding*

Nous avons défini une classe abstraite `BaseShardedCrudStorage` qui définit une interface attendue pour toutes les classes de stockage implémentant les opérations *CRUD* tout en proposant du *sharding*.

Il s'agit maintenant d'implémenter la classe `SimpleShardedCrudStorage` qui assigne les couples (clef, valeur) a un shard en utilisant le modulo du hash comme fonction de sharding (veillez à utiliser la méthode *built-in* `hash`) . Chaque shard sera une instance de la classe `SimpleCrudStorage`.



À la fin de cette étape, l'ensemble des tests doivent passer.

3 Exploration de la notion d'*IO-bound*

3.1 Introduction

Les deux classes `SimpleCrudStorage` et `SimpleShardedCrudStorage` que vous avez implémentées dans la section précédente sont utilisées par les classes `SimpleIOBoundedCrudStorage` et `SimpleIOBoundedShardedCrudStorage` que nous avons déjà implémentées. Ces deux dernières classes de stockage simulent un système qui est plus facilement limité en matière d'*IO* (entrées/sorties), ce qu'on pourrait retrouver par exemple avec de l'écriture sur un disque ou un appel réseau.

3.2 Objectifs

En adaptant le code présent dans le fichier `/key_value_part1/benchmark.py` (que vous pouvez lancer avec un `make run-benchmark`), réaliser un *benchmark* des différentes solutions de stockage (avec *sharding* ou sans, avec *IO-bound* ou sans, avec *multi-threading* ou sans — ce dernier point permet de simuler des accès "concurrents").

> Suite à ce benchmark, nous vous demandons de réaliser un mini-rapport sur les avantages et inconvénients du sharding, en utilisant la notion de *throughput*.
>
> `Python` étant un langage particulièrement utilisé en Data-Science, nous vous demandons aussi de nous présenter la notion de concurrence dans ce langage (en abordant uniquement le *multithreading*, i.e. ne vous intéressez pas pour l'instant au *multiprocessing*). Nous nous attendons donc à voir mentionner les termes *IO-Bound* et *Global Interpreter Lock* dans votre rapport.

N.B. : Il n'est pas nécessaire de vous forcer à trouver une transition entre ces deux aspects du rapport.

Les critères d'évaluations du rapport sont visibles en section 4.3.

4 Rendus et critères d'évaluation

L'ensemble des rendus devront être fait via votre repo Gitlab nominatif. Sont attendus sur la branche `main` (seule cette branche sera utilisée pour l'évaluation) :

— Votre version du fichier `/key_value_part1/simple_crud_storage.py`,
— Votre version du fichier `/key_value_part1/benchmark.py`,
— Votre rapport en lieu est place du fichier `/report.pdf`.

NB : seuls ces fichiers seront copiés vers l'environnement d'évaluation : **ne modifiez pas d'autres fichiers, vous pourriez vous tirez une balle dans le pied.**

Critères éliminatoires Impliqueront directement un 0/20 :

— L'absence de l'un de ces 3 fichiers,
— **Un fichier qui n'est pas anonyme** (mention de votre nom, login CAS, etc).

Également, merci d'avance de ne pas essayer de contourner ou de nuire au système d'évaluation...

4.1 Date de rendu

Le **03/04/2021 à 14h00** (heure de Compiègne), votre repo Gitlab nominatif sera archivé et il ne vous sera plus possible d'y apporter des modifications.

4.2 (8pts) Code python

— (3pts) Respect du style et de la qualité du code. (i.e. l'exécution de `make style` doit se faire sans erreur.)
— (5pts) Les tests fonctionnent. Au prorata des tests qui passent.

4.3 (12pts) Rapport

Critère éliminatoire Impliqueront directement un 0/12 :

— Plus d'**1 seule page au total** dans cette configuration (notez bien la taille des marges et de la police) :

```
\documentclass[12pt]{article}
\usepackage[a4paper,margin=3cm]{geometry}
```

Critères d'évaluation

— (4pts) Clarté et honnêteté scientifique du / des graphique(s) du benchmark (échelle, légende, intervalles de confiance, etc),
— (3pts) Clarté, justesse et qualité de l'analyse du benchmark,
— (5pts) Compréhension de la concurrence en `python` basée sur des sources fiables.