



Démonstration PODR

Mathieu Campan, Besson Germain, Simon Hautesserres
Groupe L1

Département Sciences du Numérique
2022-2023

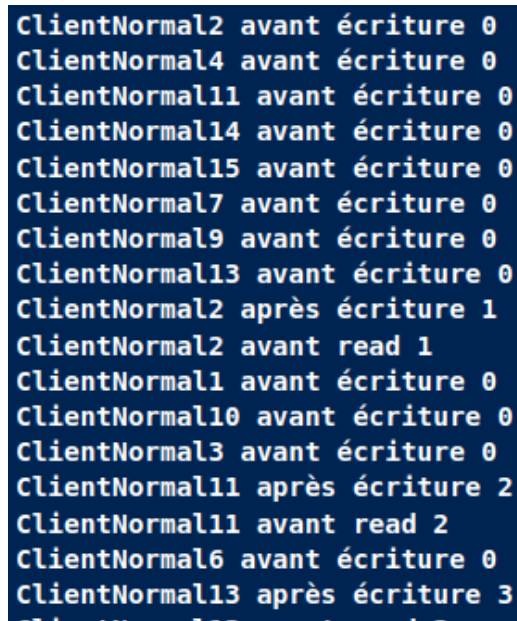
Introduction :

Ce document présente quelques scénarios d'utilisation de l'application IRC, permettant d'illustrer les comportements attendus par notre protocole.

1 Premier scénario : Défaillances aussi élevées que possible

Dans ce scénario, nous lançons notre script **lancementTest.sh** avec 15 clients normaux. Ces clients normaux (comme les Lazy) effectuent un nombre d'itérations établis dans leur fichier et alternent entre écriture et lecture. La probabilité qu'il y ait une panne sur un client est assez élevée (ici on tue un client avec 1 chance sur 10 avec 15 itérations).

Nous exécutons le script avec *\$sh lancementTest.sh* après avoir lancé le **Server.java**. Celui-ci va écrire dans le fichier test.txt (généré si il n'existe pas) les écritures et les lectures de chaque client ainsi que la version qu'ils ont au moment avant et après l'opération.

A screenshot of a terminal window with a dark blue background and white text. It displays a list of client activities. Each line shows a client ID, a status (avant écriture, après écriture, or avant read), and a value (0, 1, 2, or 3). The clients listed are ClientNormal2, ClientNormal4, ClientNormal11, ClientNormal14, ClientNormal15, ClientNormal7, ClientNormal9, ClientNormal13, ClientNormal2, ClientNormal2, ClientNormal11, ClientNormal10, ClientNormal3, ClientNormal11, ClientNormal11, ClientNormal6, and ClientNormal13. The values for 'avant écriture' are all 0. The values for 'après écriture' are 1, 2, and 3. The value for 'avant read' is 1.

```
ClientNormal2 avant écriture 0
ClientNormal4 avant écriture 0
ClientNormal11 avant écriture 0
ClientNormal14 avant écriture 0
ClientNormal15 avant écriture 0
ClientNormal7 avant écriture 0
ClientNormal9 avant écriture 0
ClientNormal13 avant écriture 0
ClientNormal2 après écriture 1
ClientNormal2 avant read 1
ClientNormal11 avant écriture 0
ClientNormal10 avant écriture 0
ClientNormal3 avant écriture 0
ClientNormal11 après écriture 2
ClientNormal11 avant read 2
ClientNormal6 avant écriture 0
ClientNormal13 après écriture 3
```

FIGURE 1 – Exemple d'affichage dans le fichier test.txt

D'autre part, si un client termine, c'est à dire qu'il a effectué toutes ses opérations, il reste actif et écrit dans le fichier test.txt qu'il a fini pour nous le signaler (d'où la nécessité du *\$killall java* après chaque lancement. Il indiquera également si il meurt. Grâce à cela, nous pouvons compter le nombre de clients ayant fini leurs opérations et ceux qui sont morts avant de finir. Dans ce scénario, nous avons eu :



FIGURE 2 – Nombre de clients morts pour les 15 itérations



FIGURE 3 – Nombre de clients ayant fini leurs tâches pour 15 itérations

Nous voyons donc que pour ce scénario 11 clients sont tombés en panne avant la fin et seulement 4 clients ont pu finir correctement leurs tâches. Cela représente largement plus de la moitié des serveurs qui sont tombés en panne pourtant les versions restent cohérentes malgré tout.

```
ClientNormal10 avant read 56
ClientNormal10 après read 56
ClientNormal10 avant écriture 56
ClientNormal11 après écriture 57
ClientNormal11 avant read 57
ClientNormal10 après écriture 58
ClientNormal11 après read 58
ClientNormal11 avant écriture 58
ClientNormal10 a fini
ClientNormal8 après écriture 59
ClientNormal8 avant read 59
ClientNormal3 après read 52
ClientNormal3 avant écriture 52
ClientNormal11 après écriture 60
ClientNormal11 a fini
ClientNormal8 après read 61
ClientNormal8 avant écriture 61
ClientNormal3 après écriture 61
ClientNormal3 avant read 61
ClientNormal3 après read 62
ClientNormal3 avant écriture 62
ClientNormal8 après écriture 62
ClientNormal8 a fini
ClientNormal3 après écriture 63
ClientNormal3 avant read 63
ClientNormal3 après read 63
ClientNormal3 avant écriture 63
ClientNormal3 après écriture 64
ClientNormal3 a fini
```

FIGURE 4 – Cohérence des résultats malgré une panne consécutive à la fin

Nous avons donc confirmé le bon fonctionnement de notre protocole pour un nombre de pannes consécutives des clients.

Cependant, nous remarquons que certains clients se bloquent et ne finissent pas lors de certaines exécutions parce que lorsque plus de la moitié des clients encore en vie meurent simultanément, le client précédent n'obtiendra jamais la moitié des réponses si il était en train de faire une enquête. (ce risque augmente vers la fin de l'exécution car il y a moins de clients donc nous atteignons plus rapidement cette moitié de clients).

2 Deuxième scénario : Défaillances plus faibles et plus d'Itérations

Dans ce scénario, nous lançons notre script `lancementTest.sh` avec 10 clients normaux. La probabilité qu'il y est une panne sur un client est plus faible (ici on tue un client avec 5 chance sur 100) mais nous avons grandement augmenté le nombre d'itération : 50 itérations. Ce scénario représente le cas optimale pour notre projet, et pour notre stratégie d'enquête (**voir rapport**). En effet, avec des clients tombant en panne de façon sporadique, notre stratégie a le temps d'évoluer sans bloquer le client.

```
ClientNormal5 après écriture 78
ClientNormal5 avant read 78
ClientNormal11 après read 77
ClientNormal11 a fini
ClientNormal5 après read 78
ClientNormal5 avant écriture 78
ClientNormal5 après écriture 79
ClientNormal5 avant read 79
ClientNormal5 après read 79
ClientNormal5 avant écriture 79
ClientNormal5 après écriture 80
ClientNormal5 avant read 80
ClientNormal5 après read 80
ClientNormal5 avant écriture 80
ClientNormal5 après écriture 81
ClientNormal5 avant read 81
ClientNormal5 après read 81
ClientNormal5 avant écriture 81
ClientNormal5 après écriture 82
ClientNormal5 avant read 82
ClientNormal5 après read 82
ClientNormal5 avant écriture 82
ClientNormal5 après écriture 83
ClientNormal5 avant read 83
ClientNormal5 après read 83
ClientNormal5 avant écriture 83
ClientNormal5 après écriture 84
ClientNormal5 avant read 84
ClientNormal5 après read 84
ClientNormal5 a fini
```

FIGURE 5 – ClientNormal5 arrive à finir en étant seul

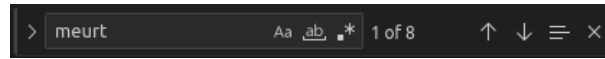


FIGURE 6 – Nombre de clients morts pour les 50 itérations

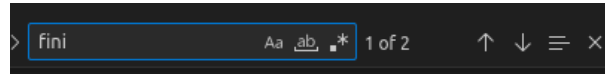


FIGURE 7 – Nombre de clients ayant fini leurs tâches pour 50 itérations

Nous observons que chaque client arrive à terminer, ou meurt mais ne bloque pas. En effet, il y a peu de chance dans ce scénario que plus de 50% des clients meurent simultanément entraînant un fonctionnement optimal de notre projet.

3 Troisième scénario : inversion de valeurs pour les registres réguliers

Nous avons voulu tester dans ce scénario le problème illustré dans le sujet ci-dessous pour les registres réguliers :

Remarque : un registre régulier tolère des exécutions « contre-intuitives », où il est possible d’observer des « inversions de valeur ». Ainsi dans l’exemple suivant, le lecteur pourrait observer successivement que le registre R prend les valeurs 2, puis 1, puis 2, ce qui ne correspond pas à l’ordre des écritures réalisées au niveau de l’écrivain (0, puis 1, puis 2).

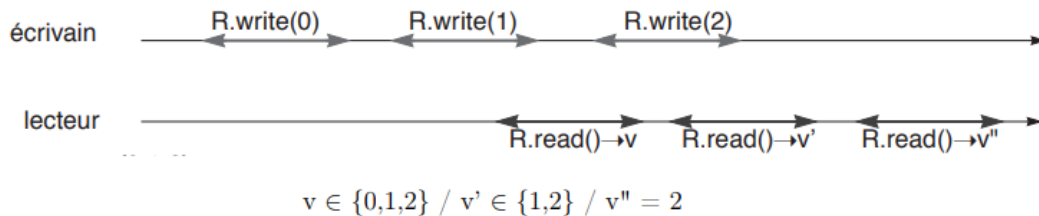


FIGURE 8 – Exemple changement de registre du sujet

Pour observer l’inversion de valeurs possible pour les registres réguliers nous avons utilisé comme précédemment le script `$sh lancementTest.sh`. Nous avons configuré le script avec 3 ClientNormal. Le clientNormal numéro 1 est l’unique écrivain et tous les autres sont des lecteurs.

Nous lançons donc le script de test de la manière suivante :

```

simonh@simonh-VirtualBox:~/Documents/2A/Projet_Donnees_Reparties/S8/Projet_Donnees_Reparties_S8$ sh lancementTest.sh
!== Server.java doit être lancé avant de lancer ce script ==!
!== Pensez à actualiser le nombre de clients totaux dans Server.java si il y a une erreur ==!
!== Pensez à faire un killall java pour tuer tous les processus en fond ==!
!== Pensez à modifier la probabilité de panne dans les clients ==!

== LANCEMENT DU TEST AVEC 3 CLIENTS NORMAUX ET 0 CLIENTS LAZY ==

==> Résultats en cours dans test.txt ...

```

FIGURE 9 – Lancement du test avec 3 clients normaux : un écrivain et deux lecteurs

Nous obtenons les résultats suivants dans le fichier test.txt

```

ClientNormal1 après écriture 16
ClientNormal1 avant écriture 16
ClientNormal3 après read 13    <= Dernière version obtenue égale à 13
ClientNormal3 avant read 13    <= Il commence une nouvelle lecture
ClientNormal3 après read 11    <= Il obtient la version 11 == Inversion de valeurs
ClientNormal1 après écriture 17
ClientNormal1 avant écriture 17

```

FIGURE 10 – Inversion de valeurs

Nous pouvons observer que le client normal 3 en lecture a terminé sa lecture précédente avec une nouvelle version valant 13. Même si la dernière écriture est à la version 16, il est normal qu'il ne l'est pas forcément reçu à cause du temps de propagation de la mise à jour. Il commence une nouvelle lecture avec cette 13ième version et finit sa lecture avec la version 11. Il obtient donc une version antérieure à celle qu'il avait.

D'autre part, si on augmente le nombre de clients normaux à 10, nous retrouvons bien ce phénomène :

```

ClientNormal7 après écriture 7
ClientNormal7 avant read 7 <== Commence lecture avec sa version écrite 7

```

FIGURE 11 – Début inversion de valeurs

```

ClientNormal9 après écriture 13
ClientNormal9 avant read 13
ClientNormal7 après read 4 <== Termine lecture avec une autre version écrite 4

```

FIGURE 12 – Fin inversion de valeurs

En effet, ClientNormal7 finit d'écrire avec la version 7 qu'il a en mémoire en commençant sa lecture. Mais il finit sa lecture avec une version antérieure à celle qu'il avait pourtant lui même modifié.

Nous avons donc bien dans les deux cas une inversion de valeurs pour les registres réguliers qui sera corrigée dans la partie registre atomique ci-dessous.

4 Quatrième scénario : absence d'inversion de valeurs pour les registres atomiques

Pour tester le registre atomique, nous avons utilisé le script précédent **lancementTest.sh**. Nous l'avons lancé avec 10 ClientNormal. Chaque client peut être écrivain ou lecteur en même temps. Nous avons défini la probabilité de panne à 0 car ce paramètre a déjà été abordé dans la partie précédente. Comme précédemment, nous exécutons le script après avoir lancé le **Server.java**.

Les résultats obtenus sont écrits dans le fichier test.txt.

```
ClientNormal1 après read 13
ClientNormal1 avant écriture 13
ClientNormal2 après écriture 17
ClientNormal2 avant read 17
ClientNormal9 après écriture 16
ClientNormal9 avant read 16
ClientNormal1 après écriture 18
ClientNormal1 avant read 18
ClientNormal10 après read 14
ClientNormal10 avant écriture 14
ClientNormal8 après écriture 19
ClientNormal8 avant read 19
ClientNormal10 après écriture 20
ClientNormal10 avant read 20
ClientNormal7 après écriture 21
ClientNormal7 avant read 21
ClientNormal4 après read 14
ClientNormal4 avant écriture 14
ClientNormal6 après read 15
ClientNormal6 avant écriture 15
ClientNormal4 après écriture 22
ClientNormal4 avant read 22
ClientNormal10 après read 20
ClientNormal10 avant écriture 20
ClientNormal8 après read 21
ClientNormal8 avant écriture 21
ClientNormal9 après read 16
ClientNormal9 avant écriture 16
ClientNormal6 après écriture 23
ClientNormal6 avant read 23
```

FIGURE 13 – Registre atomique sans inversion de valeurs

Grâce au Ctrl-F, nous avons pu vérifier chaque écriture et lecture des clients un à un, et nous remarquons qu'ils ne rencontrent pas d'inversion de valeurs avec le registre atomique contrairement au registre régulier.