INP ENSEEIHT

Systèmes concurrents

2ème année Sciences du Numérique

TP : Transactions

12 décembre 2022

1 Thèmes

— exemples et scénarios de programmation concurrente transactionnelle
— problèmes de cohérence induits par la mémoire transactionnelle
— mise en œuvre de différents protocoles de propagation et de contrôle de concurrence
— évaluation des performances et de l'intérêt de différents protocoles de contrôle de concurrence, selon le contexte d'exécution et le profil des applications.

2 Préparation

Le TP propose un service de mémoire transactionnelle, permettant d'implanter différentes politiques de propagation des écritures et de contrôle de concurrence.

La mémoire transactionnelle proposée se présente comme un ensemble d'objets. Chaque objet de cette mémoire transactionnelle possède deux attributs : un identifiant (String) et une valeur (int).

Le service de mémoire transactionnelle contrôle l'exécution d'un ensemble de transactions définies par une classe abstraite (*AbstractTM*), et lancées par appel à la méthode *newTransaction* de cette interface. Chaque transaction effectue une série d'accès, et s'achève par une validation ou un abandon.

Un interpréteur interactif (*shell*) permet de lancer et d'exécuter pas à pas un ensemble de transactions sur ce service de mémoire transactionnelle. Ces transactions peuvent être contrôlées pas à pas ou spécifiées par un script. Les commandes de cet interpréteur sont les suivantes :

— Opérations transactionnelles

new <transaction> lancer une transaction d'identifiant <transaction>

<transaction> commit valider une transaction

<transaction> abort abandonner une transaction

<transaction> read <t_objet> lire un objet

<transaction> write <t_objet> <valeur> affecter une valeur à un objet

— Opérations du shell

init <tm> (t_object,valeur)* choisit un service de mémoire transactionnelle <tm> et en définit l'ensemble des objets transactionnels avec leur valeur ;

run <script> exécute ligne à ligne les commandes contenues dans un fichier script

list liste le contenu de la mémoire transactionnelle

help aide en ligne

exit quitte le simulateur

Par ailleurs, **un mode simulateur** permet d'évaluer l'efficacité des différents services de mémoire transactionnelle sur des scénarios d'exécution qui peuvent lui être fournis. Ce mode simulateur est présenté en section 4.

L'archive fournie contient un fichier *README* qui fournit une aide (minimale) à l'utilisation de ces outils.

3 Exemples et scénarios d'exécution

Note : dans ce qui suit, il est très largement préférable de réfléchir à une réponse **avant** de chercher à l'obtenir par l'observation d'une exécution au moyen du shell.

1. On considère le script/scénario suivant, où, pour des raisons de lisibilité, on a aligné dans une même colonne les opérations relatives à une même transaction, et où XXX doit être remplacé par le nom du protocole choisi : $TMPP$, $TMPC\ldots$ (voir ci-dessous) :

```
init XXX (x,0) (y,0) (z,0) (ft1,0) (ft2,0) (ft3,0)
```

	T_1	T_2	T_3
(1)	new T1		
(2)		new T2	
(3)			new T3
(4)		T2 write y 1	
(5)	T1 read y		
(6)			T3 read z
(7)		T2 write z 2	
(8)	T1 read x		
(9)		T2 read x	
(10)			T3 write x 3
(11)	T1 write ft1 1		
(12)	T1 commit		
(13)		T2 write ft2 1	
(14)		T2 commit	
(15)			T3 write ft3 1
(16)			T3 commit

temps ↓

```
list
```

On dispose de trois protocoles de contrôle de concurrence [1] : $TMPP$ réalisant une ***politique de contrôle de concurrence pessimiste*** sans blocage et avec ***propagation directe*** (protocole A dans la section 5), $TMPC$ réalisant une ***politique de contrôle de concurrence par certification*** avec ***propagation différée***, et $TM2PL$ réalisant le protocole de verrouillage à deux phases [2] avec propagation directe.

Quel sera le résultat de cette exécution selon les différentes politiques utilisées :

(a) Quelles seront les transactions validées, bloquées ou abandonnées ?

(b) Quel sera l'ordre série équivalent obtenu ?

(c) Quelles seront les valeurs de variables affichées par la commande `list` ?

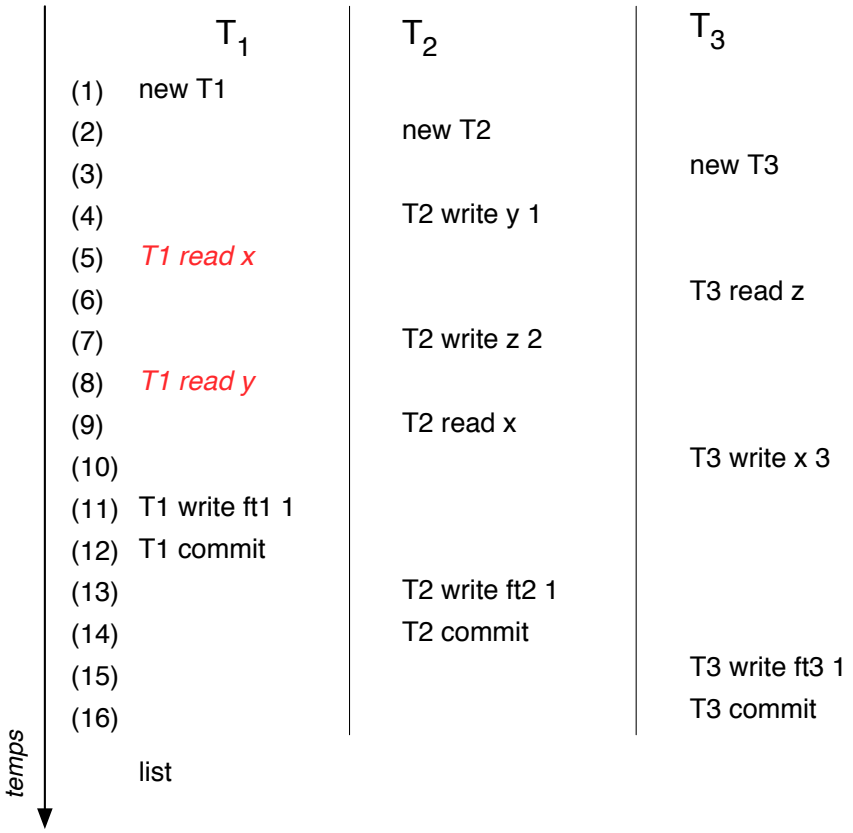
(d) Vérifiez vos réponses à l'aide du shell.

Note : un scénario pour $TMPP$ est fourni dans l'archive à titre d'exemple.

1. Un glossaire situé en annexe rappelle la définition des termes en ***italique gras***
2. Voir le cours : le glossaire n'est tout de même pas un rappel de cours...

2. Mêmes questions pour le scénario suivant :

init XXX (x,0) (y,0) (z,0) (ft1,0) (ft2,0) (ft3,0)



3. On utilise maintenant une politique d'accès sans contrôle de concurrence ($TMNoCC$) et avec propagation directe. Illustrer la limite de la propagation directe sur un scénario simple, interprété par le shell.

4 Évaluation de protocoles

Le simulateur fourni permet d'évaluer et de comparer les temps d'exécution et le bon achèvement d'un ensemble de transactions concurrentes exécutées suivant différents services de mémoire transactionnelle. Les transactions simulées pourront être de deux types :

T (Transaction), qui correspond à une transaction standard ;

S (Super transaction), qui consiste à itérer une transaction simple jusqu'à ce qu'elle aboutisse (valide).

Chaque exécution est spécifiée par un scénario. Les scénarios sont définis dans un fichier de texte dont la structure est la suivante :

— une ligne d'en-tête spécifiant le protocole à évaluer et la liste des variables de la mémoire.

— une ligne pour chaque transaction. Chaque ligne commence par l'identifiant de la transaction, suivi du type de la transaction (T ou S), suivi de la séquence des opérations de la transaction, séparées par des points-virgule.

— les opérations possibles et leur syntaxe sont les suivantes :

commit valider ;
abort abandonner ;
write x écrire la variable x ;
read x lire la variable x ;
process n effectuer un calcul interne (sans accès aux variables transactionnelles) de n unités de temps.
Note : les valeurs lues ou écrites ne sont pas précisées, étant inutiles à l'évaluation de performances.

Le simulateur considère que l'exécution des opérations autres que *process* (hors temps de blocage) prend une unité de temps.

La simulation se termine lorsque toutes les transactions sont terminées, ou lorsque toutes les transactions non terminées sont bloquées. Le simulateur produit alors un rapport précisant :

— la durée totale (en unités de temps) de l'exécution concurrente ;

— l'ensemble des transactions validées, abandonnées, bloquées, ainsi que, pour les transactions non validées, le point de l'exécution atteint par la transaction (dernière opération effectuée) ;

— pour chaque transaction, le temps total consommé, le temps utile (calculs effectués par les transactions validées), le temps improductif (abandon). Le temps total consommé est la somme du temps utile et du temps improductif.

— le degré de concurrence (somme des temps/durée totale) pour les transactions validées (à comparer au nombre de transactions validées). Cette métrique est pertinente si toutes les transactions ont validé.

Questions

1. Commenter les scénarios fournis ;

```
TM2PL x y z
t1 T : process 5 ; read y ; process 10 ; read x ; process 8 ; commit
t2 T : process 2 ; write y ; process 10 ; write z ; process 5 ; read x ; process 10 ; commit
t3 T : process 8 ; read z ; process 12 ; write x ; process 10 ; commit

TMPC x y z
t1 T : process 5 ; read y ; process 10 ; read x ; process 8 ; commit
t2 T : process 2 ; write y ; process 10 ; write z ; process 5 ; read x ; process 10 ; commit
t3 T : process 8 ; read z ; process 12 ; write x ; process 10 ; commit

TMPC x y z
t1 S : process 5 ; read y ; process 10 ; read x ; process 8 ; commit
t2 S : process 2 ; write y ; process 10 ; write z ; process 5 ; read x ; process 10 ; commit
t3 S : process 8 ; read z ; process 12 ; write x ; process 10 ; commit
```

2. Définir des scénarios permettant de caractériser les situations favorables à chacune des politiques fournies.

5 Mise en œuvre des protocoles de contrôle de concurrence

Cette section présente les structures de données et algorithmes de principe des services de mémoire transactionnelle proposés dans le cadre de ce TP.

Verrous

Les protocoles de mémoire transactionnelle proposés s'appuient sur un service de verrous. Un verrou est un objet L sur lequel les opérations suivantes sont possibles :

- **lock-shared** $L.lock\_shared()$ demande à d'acquérir L en mode partagé et retourne lorsque l'acquisition est réussie ;
- **trylock-shared** $L.trylock\_shared()$ demande à d'acquérir L en mode partagé et retourne *sans blocage* vrai si l'acquisition est réussie et faux sinon ;
- **lock** $L.lock()$ tente d'acquérir L en mode exclusif et retourne lorsque l'acquisition est réussie ;
- **trylock** $L.trylock()$ tente d'acquérir L en mode exclusif et retourne *sans blocage* vrai si l'acquisition est réussie et faux sinon ;
- **unlock** $L.unlock()$ libère L.

En mode partagé, un verrou peut être acquis simultanément par plusieurs transactions. En mode exclusif, un verrou ne peut être obtenu que par une transaction à la fois. Les modes partagé et exclusif ne sont pas compatibles.

Mémoire transactionnelle

Les protocoles de mémoire transactionnelle proposés implémentent une interface commune. Cette interface définit

— les opérations de lecture et d'écriture sur une mémoire partagée vue comme un tableau $Mem[0..Max]$ de mots mémoire. Un mot de la mémoire partagée Mem est considéré comme un registre R sur lequel deux opérations sont possibles :

- **R.read(T)** qui fournit à la transaction T le contenu courant du mot mémoire R
- **R.write(T,val)** qui permet à la transaction T d'affecter la valeur val au mot mémoire R

— les opérations de création et terminaison des transactions, notamment :

- **commit(T)** qui termine la transaction T et valide ses effets ;
- **abort(T)** qui termine la transaction T et annule ses effets.

Les protocole de mémoire transactionnelle utilisent les structures de données suivantes :

— $Mem[0..Max]$, la mémoire partagée vue comme un tableau de mots mémoire ;

— $L[0..Max]$, un tableau de verrous. A chaque mot mémoire $Mem[i]$ est associé un verrou $L[i]$ qui en contrôle le partage. Initialement, tous les verrous sont libres.

Chaque transaction T peut utiliser des attributs privés (locaux, non partagés) :

— Un tableau $SvMem[0..Max]$ de mots mémoire contenant, pour les mots de Mem accédés par T, la valeur du mot avant son premier accès par T

— Un tableau $ResMem[0..Max]$ de mots mémoire contenant, pour les mots de Mem accédés par T, la valeur du mot après son dernier accès par T

— Un ensemble lus contenant les indices des mots de Mem lus par T. Cet ensemble est initialement vide.

— Un ensemble $ecrits$ contenant les indices des mots de Mem écrits par T. Cet ensemble est initialement vide.

Protocole A

Les opérations de ce protocole sont les suivantes :

— operation m .$read(T)$
if $m \notin T.lus \cup T.ecrits$ then
// *tenter d'obtenir* $L[m]$ *en mode partage*
if not $L[m].trylock\_shared()$ then
$abort(T)$;
endif ;
$T.lus := T.lus \cup \{m\}$;
endif
return $Mem[m].read()$;

— operation m .$write(T, val)$
if $m \notin T.ecrits$ then
// *tenter d'obtenir* $L[m]$ *en mode exclusif*
if not $L[m].trylock()$ then
$abort(T)$;
endif ;
$T.ecrits := T.ecrits \cup \{m\}$;
$T.SvMem[m] := Mem[m].read()$;
endif
$Mem[m].write(val)$;

— operation $commit(T)$
$unlock\_all(T)$;

— operation $abort(T)$
// *restaurer les valeurs ecrites*
foreach $m \in T.ecrits$ do $Mem[m].write(T.SvMem[m])$;
$unlock\_all(T)$;

— function $unlock\_all(T)$
// *liberer tous les verrous obtenus par* T
foreach $m \in T.lus \cup T.ecrits$ do $L[m].unlock()$;
$T.lus := \emptyset$;
$T.ecrits := \emptyset$;

Protocole B

Ce protocole utilise un verrou supplémentaire *mutex* destiné à assurer l'exclusion mutuelle sur les opérations de validation, nécessaire pour assurer la validité du test de sérialisabilité. Les opérations de ce protocole sont les suivantes :

— operation m .$read(T)$
if $m \notin T.lus$ then
if $m \notin T.ecrits$ then
$T.SvMem[m] := Mem[m].read()$
$T.ResMem[m] := Mem[m].read()$
endif
$T.lus := T.lus \cup \{m\}$;
endif
return $T.ResMem[m]$;

— operation $m.write(T, val)$
if $m \notin T.ecrits$ then
  if $m \notin T.lus$ then
    $T.SvMem[m] := Mem[m].read()$;
  endif
  $T.ecrits := T.ecrits \cup \{m\}$;
endif
$T.ResMem[m] := val$;

— operation $commit(T)$
$mutex.lock()$;
foreach $m \in T.lus$ do $ok := ok$ and $(T.SvMem[m] = Mem[m].read())$;
if ok then
  foreach $m \in T.ecrits$ do $Mem[m].write(T.ResMem[m])$;);
else
  $abort(T)$;
endif
$mutex.unlock()$;

— operation $abort(T)$
return;

Questions

1. Indiquer (et justifier) pour chacun de ces protocoles
   - s'il est optimiste ou pessimiste
   - s'il assure la ***sérialisabilité***
   - si la propagation des valeurs est immédiate ou différée
   - s'il présente un risque d'interblocage
   - s'il présente un risque de rejet en cascade
2. L'archive fournie propose une implémentation pour chacun des protocoles A et B (Classes `TMPP` et `TMPC`). Définir et exécuter des scénarios à partir de l'interpréteur de commandes afin de vérifier que ces protocoles ont bien les propriétés déterminées dans la question précédente.
3. Implémenter (et tester) un protocole de contrôle de concurrence (minimal) par verrouillage (bloquant) ***à deux phases strict***.

   Indications :
   - il est possible (voire recommandé) de s'inspirer des implémentations fournies.
   - Le service de mémoire transactionnelle est défini par une interface (`TMInterface`) proposant les opérations suivantes :

     newTransaction(transaction) crée et lance une nouvelle transaction ;

     read(id_transaction,objet) renvoie la valeur courante d'un objet ;

     write(id_transaction,objet,valeur) affecte une valeur à un objet ;

     abort(id_transaction) abandonne une transaction ;

     commit(id_transaction) valide une transaction.

     Cette interface est complétée par une classe abstraite (classe `TMAbstract`) qui implémente `TMInterface` et spécifie les accès aux objets de la mémoire transactionnelle.

Glossaire

optimiste qualifie une politique privilégiant l'exécution immédiate et reportant les vérifications dans le but de favoriser le parallélisme/la vitesse de traitement.
Inconvénient : nécessite de gérer les retours en arrière dans le cas où l'exécution s'avère invalide ou abandonnée.

pessimiste qualifie une politique privilégiant la sûreté : une opération n'est exécutée que s'il est certain que son effet maintient un état global cohérence.
Inconvénient : les retours en arrière sont évités mais le parallélisme/la vitesse de traitement sont pénalisés par les mécanismes mis en place pour garantir la sûreté des opérations.

Attention : les protocoles de mémoire transactionnelles combinent deux mécanismes **différents** :

- la politique de ***propagation*** des valeurs écrites qui contrôle l'instant où une écriture est réellement effectuée en mémoire partagée
- la politique de ***contrôle de concurrence*** qui contrôle l'ordonnancement des opérations de manière à garantir la sérialisabilité.

Chacune de ces deux politiques peut être optimiste ou pessimiste **indépendamment de l'autre.** (Certaines combinaisons sont cependant plus appropriées que d'autres).

(politique de) contrôle de concurrence optimiste les transactions ne sont pas contrôlées en cours d'exécution mais la sérialisabilité est vérifiée au moment de la validation et aboutit à la décision d'accepter définitivement les résultats ou de les annuler complètement.

(politique de) contrôle de concurrence par certification voir (politique de) contrôle de concurrence optimiste

(politique de) contrôle de concurrence pessimiste la sérialisabilité est testée à chaque accès et aboutit soit à la poursuite de la transaction, soit à son blocage, soit à son rejet.

propagation différée les écritures de la transaction ne se font pas sur la mémoire partagée mais sur une copie privée (espace de travail/journal après) qui est recopiée en mémoire partagée en cas de validation.

propagation directe les écritures de la transaction se font directement sur la mémoire partagée.

sérialisabilité l'exécution concurrente d'un ensemble de transactions est sérialisable s'il existe une exécution en série (l'une après l'autre) de ces transactions qui aboutit au même résultat, c'est-à-dire au même état de la mémoire transactionnelle.

verrouillage à deux phases strict politique de verrouillage à deux phases où la phase de libération des verrous est reportée au moment de la validation (ou de l'abandon).