SCIENCES SORBONNE UNIVERSITÉ

Projet ARA 2022–2023

Jonathan Lejeune

Objectifs

Le but de ce projet est de faire une étude expérimentale à travers le simulateur PeerSim. Dans un premier temps, nous étudierons dans un environnement fiable une application fictive qui se base sur un protocole de verrouillage distribué. Dans un second temps, nous exécuterons cette application dans un environnement non fiable (un nœud peut tomber en panne). Dans ce cas, l'application se basera sur un protocole distribué de points de reprise (checkpointing) permettant de sauvegarder l'état des nœuds régulièrement et de reprendre l'exécution à un état stable en cas d'erreur ou de panne.

Consignes générales

- Ce travail est à faire seul ou (de préférence) en binôme. Les trinômes ne sont pas autorisés.
- Les livrables sont à rendre sur Moodle avant la date butoire indiquée
- Il est attendu de rendre 2 fichiers :
  - ◇ Une archive de vos **sources Java** (éviter les archives trop volumineuses, en évitant par exemple d'inclure les fichiers .class)
  - ◇ Votre rapport, au format PDF
- Pour chaque étude expérimentale, vous devrez :
  - ◇ porter une attention particulière à la présentation graphique de vos résultats et à leur lisibilité.
  - ◇ vérifier que vos résultats ne sont pas aberrants en évaluant au préalable la forme attendue des courbes.
  - ◇ commenter et interpréter vos résultats
  - ◇ synthétiser vos résultats par une conclusion.

Préparation du projet

Créez un nouveau projet Eclipse, configurez le build path pour prendre en compte les jar de peersim (voir TP) et y copiez les sources java associées à ce sujet. Assurez vous que la compilation se passe bien (absence de croix rouges).

Dans les sources vous trouverez :

- le package `ara.projet.mutex` qui comporte les classes suivantes :

    - ◇ `NaimiTrehelAlgo` : une classe implantant une variante de l'algorithme d'exclusion mutuelle de Naimi-Trehel.
    - ◇ `InternalEvent` : classe décrivant un appel de primitive sur l'algorithme de mutex.
    - ◇ `NaimiTrehelAlgoCheckpointable` : une classe étendant la classe `NaimiTrehelAlgo` précédente pour pouvoir s'exécuter dans un environnement non fiable à l'aide d'un protocole de point de reprise sous-jacent.
- le package `ara.projet.checkpointing` qui comporte les classes suivantes :
    - ◇ `Checkpointable` : une interface décrivant les méthodes que doit offrir une application s'exécutant sur un protocole de point de reprise. Cette interface est par exemple implantée par la classe `NaimiTrehelAlgoCheckpointable`.
    - ◇ `Checkpointer` : une interface décrivant les méthodes qu'un protocole de point de reprise doit implanter.
    - ◇ `NodeState` : une classe permettant de décrire un état à sauvegarder lors d'un point de reprise.
    - ◇ `CrashControler` : un contrôleur peersim permettant de simuler la panne d'un ou plusieurs nœuds et de démarrer en conséquence un mécanisme de recouvrement
- le package `ara.projet.checkpointing.algorithm` qui contient une classe décrivant un protocole de points de reprise/recouvrement
- le package `ara.projet.util` contenant les classes suivantes
    - ◇ `Constantes` ne contenant que des champs statiques. On notera la présence d'un logger pour gérer les affichages produits par le code de la simulation.
    - ◇ `FIFOTransport` : protocole de transport assurant des communications FIFO (voir TP d'initiation Peersim)
    - ◇ `Message` : classe décrivant un message général avec les entêtes requises (voir TP d'initiation Peersim).

Analyse de l'application

Principe de l'application

Le principe de l'application repose sur un accès périodique des nœuds d'un système distribué à une ressource partagée. Pour synchroniser les accès à cette ressource, on utilise un protocole de verrouillage distribué. Chaque nœud exécute ainsi périodiquement les étapes suivantes :

1. prendre le verrou,
2. entrer en section critique (appelée CS par la suite) : dans notre cas, ceci consiste à l'incrémentation d'un compteur partagé et d'attendre une certaine période
3. relâcher le verrou

Le protocole de verrouillage est implanté avec un algorithme distribuée d'exclusion mutuelle.

Exclusion mutuelle

Un algorithme d'exclusion mutuelle doit respecter les deux propriétés suivantes :

- **Sûreté** : il doit y avoir au plus un nœud en CS
- **Vivacité** : en supposant des temps d'exécution de CS finis alors tout nœud demandant l'accès à la CS doit l'obtenir dans un temps fini (pas de famine).

L'algorithme que l'on considère ici est une variante de l'algorithme de Naimi-Tréhel qui est un algorithme basé sur la circulation d'un jeton. Le jeton est un élément logique que les nœuds du système se transmettent. Seul le détenteur du jeton peut entrer en CS. L'unicité du jeton assure donc la propriété de sûreté. Si ce dernier représente la possession d'accès à la CS, il peut aussi contenir les données partagées qui ne sont accessibles que de manière exclusive par les nœuds. Ici le compteur partagé incrémenté à chaque exécution de CS sera stocké dans le jeton. Le jeton peut être dans trois états possibles :

- **Utilisé** : le nœud porteur du jeton est en section critique
- **en Transit** : le jeton n'est possédé par aucun nœud mais est en transit dans le réseau
- **Non utilisé** : le nœud porteur du jeton ne l'utilise pas.

Les transitions entre les différents états ont donc la sémantique suivante :

- U $\rightarrow$ T : le nœud sort de la section critique et transmet le jeton au nœud suivant
- T $\rightarrow$ U : un nœud reçoit le jeton et entre section critique
- U $\rightarrow$ N : le nœud sort de CS mais il n'existe aucun autre nœud qui souhaite entrer en CS
- N $\rightarrow$ U : le nœud porteur du jeton entre en CS
- T $\rightarrow$ N : Le cas où un nœud reçoit le jeton alors qu'il ne l'a pas demandé ne peut pas se produire dans l'algorithme de Naimi-Tréhel
- N $\rightarrow$ T : le nœud porteur du jeton n'est pas en section critique et transmet le jeton à un autre nœud qui le demande

La variante considérée permet de remplacer la file distribuée *next* par un ensemble de files locales formant logiquement une file distribuée. L'article de cette variante étant relativement difficile à trouver sur le web, une copie de l'article est disponible dans les ressources de ce projet. Dans l'algorithme de Naimi-Trehel, on considère que le graphe de communication est complet, c'est à dire que tout nœud peut envoyer directement un message à n'importe quel autre nœud.

Dans cet exercice nous nous concentrerons uniquement sur l'application concernée sans encore tenir compte des mécanismes de sauvegarde et de restauration des points de reprises qui seront traités aux exercices suivants. Nous considérons donc que le système et le réseau sont fiables (ni panne de nœud, ni perte ou duplication de message).

Travail demandé : étude expérimentale 1

Scénario considéré : Chaque nœud respecte le principe de l'application en demandant périodiquement la section critique. On supposera que les nœuds ne peuvent pas initier une nouvelle demande de CS tant que la précédente n'a pas été satisfaite. Autrement dit un nœud ne peut faire au plus qu'une demande de section critique à la fois. Nous considérons que le système et le réseau sont fiables : ni panne de nœud, ni perte ou duplication de message).

Métriques considérées dans cette étude :

- le nombre de messages applicatifs (`token` et `request`) par section critique

- le nombre de messages `request`) par nœud
- le temps nécessaire moyen pour obtenir la section critique, c'est à dire le temps moyen qu'un nœud passe dans l'état `requesting`.
- le pourcentage de temps que le jeton passe dans chacun de ses états (U, T et N).

Paramètres :
- le nombre de nœuds noté N (paramètre du simulateur)
- le temps moyen passé en section critique, noté α (paramètre du protocole applicatif)
- le temps moyen qu'un nœud attend avant de redemander l'accès à la section critique, noté β (paramètre du protocole applicatif)
- le temps moyen pour transmettre un message entre deux nœuds, noté γ (paramètre du protocole de transport)

On note le ratio $\rho = \frac{\alpha+\gamma}{\beta}$ qui représente la charge sur la section critique. Plus la valeur de ce ratio est élevée, plus la taille de la file d'attente sur la section critique est grande.

Objectif : Quantifier l'impact de la charge sur les métriques décrites précédemment. Pour faire varier la charge, on propose de fixer N, α et γ et de faire juste varier β. Cependant, nous souhaitons étudier trois cas de figure :
- cas où la valeur de γ est négligeable par rapport à celle de α
- cas où la valeur de γ est comparable à celle de α
- cas où la valeur de γ est nettement supérieure à celle de α

Vers la considération d'un système non fiable

Nous allons maintenant supposer que les nœuds peuvent tomber en panne franche. Ceci a pour conséquence :
- de perdre le jeton si le nœud fautif était en CS (perte de la sûreté)
- de perdre des requêtes si le nœud fautif avait une file locale non vide ou si il avait des fils dans l'arbre des *last* (perte de la vivacité)

Pour limiter les conséquences qu'aurait une telle panne sur l'application on souhaite ajouter un mécanisme de points de reprise. Ainsi, l'état applicatif est sauvegardé régulièrement et un mécanisme de recouvrement sera mise en place lors de la panne d'un nœud.

Sauvegarde applicative

Nous fournissons dans le package `ara.projet.mutex` la classe `NaimiTrehelAlgoCheckpointable` qui étend la classe `NaimiTrehelAlgo` en ajoutant les propriétés d'un protocole `Checkpointable` c'est à dire les méthodes suivantes :
- NodeState getCurrentState(); qui renvoie l'état courant du nœud
- void restoreState(NodeState); qui permet de restaurer un état précédent
- void suspend(); qui permet de suspendre l'exécution du protocole pour démarrer une phase de recouvrement d'état
- void resume(); qui permet de reprendre l'exécution du protocole lorsqu'une phase de recouvrement d'état est terminée

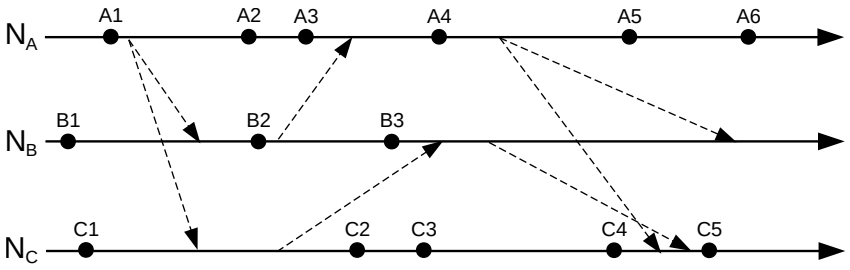
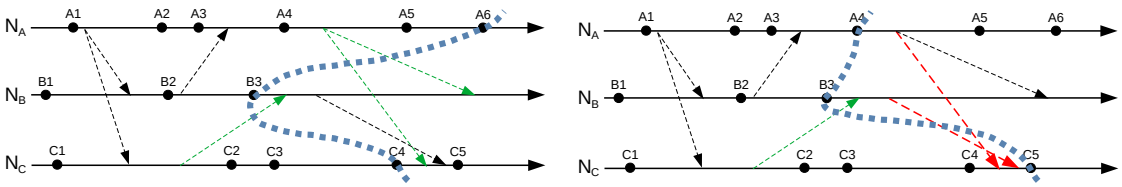


FIGURE 1 – Exécution d'un système distribué

Points de reprise et ligne de recouvrement

Un point de reprise (ou checkpoint) est une sauvegarde locale de l'état local d'un nœud. La figure 1représente sous forme de chronogramme l'exécution d'un système distribué composé de trois nœuds qui s'échangent des messages (ligne en pointillé) et qui font des points de reprise au cours du temps (représentés par un point noir sur les lignes temporelles). Si les N nœuds du système font régulièrement des points de reprise, il est ainsi possible de calculer un état global qui se caractérise par un ensemble de N points de reprise, sélectionnés sur chaque nœud du système. Un état global forme donc une ligne de recouvrement sur laquelle le système peut reprendre une exécution après une panne. Cependant, si un état global regroupe un ensemble de sauvegardes d'état local des nœuds, il doit aussi inclure les messages qui sont en transit sur le réseau. Une ligne de recouvrement doit donc être cohérente au niveau des messages en assurant que toute réception de message se trouvant avant la ligne de recouvrement ait bien sont envoi qui se trouve également avant la ligne. En revanche il est possible que l'envoi d'un message soit avant la ligne mais pas sa réception.



(a) Ligne cohérente (b) Ligne incohérente

FIGURE 2 – Exemple de ligne de recouvrement

La figure 2 expose deux exemples de ligne de recouvrement cohérente (figure 2(a)) et incohérente (figure 2(b)). Dans la version cohérente, les messages qui traversent la ligne de recouvrement sont cohérents (en vert) car leur envoi précède toujours leur réception du point de vue de la ligne de recouvrement. En revanche, dans la version incohérente, deux messages ont leur réception qui précède leur envoi du point de vue de la ligne de recouvrement.

Le but d'un algorithme de recouvrement à point de reprise est de trouver la ligne de recouvrement cohérente la plus récente dans le temps. Dans l'exemple de la figure 1, le résultat serait les points *A6*, *B3* et *C4*.

Les algorithmes à point de reprise peuvent être classifiés en deux familles : **non-coordonnée** et **coordonné**.

- Dans les algorithmes non-coordonnés, les nœuds font des points de reprise arbitrairement indépendamment des autres nœud et c'est à la charge de l'algorithme de calculer la ligne de recouvrement cohérente.
- Dans les algorithmes coordonnés, les points de reprise se font de manière synchrone par rapport aux autre nœuds en assurant que tout point de reprise est toujours cohérent par rapport aux autres. Ceci implique que la ligne de recouvrement se calcule trivialement en prenant pour tous les nœuds sont dernier point de reprise.

Description de l'implantation de l'algorithme `Juang-Venkatesan`

L'algorithme de Juang-Venkatesan est un algorithme de recouvrement à partir de point de reprises non-coordonnés. Une implantation de cet algorithme vous est fournie dans la classe `JuangVenkatesanAlgo` du package `ara.projet.checkpointing.algorithm` et l'article vous est donné dans les ressources du projet. La méthode `createCheckpoint` est appelé périodiquement sur chaque nœud via la méthode `loop`. On peut cependant remarquer que cette période n'est pas strictement fixe et égale pour tous les nœuds pour éviter d'avoir artificiellement des points de sauvegarde "alignés" dans le temps. On note aussi le fait que les points de reprise de ne se font pas nécessairement à chaque tour pour simuler encore davantage le fait que les nœud est des points de reprise non alignés dans le temps.

Les données sauvegardées dans chaque point de reprise sont

- l'état de l'application
- le nombre de messages envoyés depuis le dernier point de reprise
- le nombre de messages reçus depuis le dernier point de reprise
- les messages envoyés depuis le dernier point de reprise

La structure de donnée permettant de simuler la sauvegarde des points de reprise sur un support stable est une pile (`java.util.Stack`). À chaque création de point de reprise on empile un nouvel élément. À chaque retour en arrière (rollback) lors du calcul de la ligne de recouvrement, on dépile les points de reprise qui amènerai à une ligne de recouvrement incohérente.

On pourra remarquer que la classe est un décorateur de transport peersim. En effet, elle implante l'interface `Transport` et se base également sur un protocole de `Transport`. Ceci permet ainsi de s'insérer entre la couche supérieure (ici le protocole de mutex) et sa couche de transport pour compter les messages envoyés et reçus sans être intrusif dans le code de la couche supérieure. De plus, on utilise un message spécifique propre au protocole de checkpointing. Ces messages encapsulent tous les messages provenant de la couche supérieure. Si ces messages n'existaient pas, le simulateur pourrait délivrer les messages à la couche du dessous en court-circuitant la couche de protocole de checkpointing ce qui empêcherait d'incrémenter les compteurs de réception.

Pour démarrer une phase de recouvrement, tous les nœuds doivent appeler la méthode `recover`. La phase de recouvrement se termine à la fin de la méthode `stop_recover`. Le schéma de l'algorithme de recouvrement est le suivant :

1. Suspendre l'exécution de la couche supérieure
2. Passer à l'état `recovery`
3. Trouver une ligne de recouvrement cohérente avec l'algorithme de Juang-Venkatesan
4. Barrière de synchronisation globale

5. Retrouver les messages envoyés mais non reçus (message vert dans la figure 2)
   (a) Chaque nœud diffuse un message à tous les autres nœud pour indiquer le nombre de messages reçus
   (b) À la réception d'un tel message, un nœud est capable de savoir en faisant la différence avec son compteur `sent`, combien de messages il faut renvoyer à l'expéditeur du message
   (c) Un message de réponse avec la liste des messages à rejouer est envoyé (ce message peut être vide)
   (d) Lorsqu'un nœud a reçu ce message de tous les autres nœuds, on peut alors stopper l'algorithme de recouvrement
6. Réactiver la couche supérieure
7. Restaurer l'état de la couche supérieure, et mettre a jour l'état de la couche de checkpointing (compteurs, suppression des messages envoyés ...)
8. Rejouer les messages non reçus qui ont été récupérés à l'étape 5.

Test de l'intégration des deux protocoles

Il vous est conseillé de tester le bon fonctionnement de `NaimiTrehelAlgoCheckpointable` avec `JuangVenkatesanAlgo` Considérez 5 nœuds sur lequel vous planifierez grâce au contrôleur `CrashControler` une panne sur le nœud 4 (`probacrash` = 1) à un moment arbitraire de la simulation. Á la suite de la panne du ou des nœud(s) concernés, le contrôleur démarre sur tous les nœuds (y compris les fautifs) le protocole de recouvrement. Lorsque le recouvrement est terminé, vérifiez que l'algorithme de Checkpointing permet à l'application de revenir à un état cohérent et de rejouer son exécution (ceci peut se voir avec la valeur du compteur du jeton et/ou le compteur de section critique par nœud). N'oubliez pas que pour l'algorithme de Juang-Venkatesan les canaux doivent être FIFO et qu'il faut utiliser en conséquence le protocole de transport FIFO qui vous a été fourni.

Travail demandé : étude expérimentale 2

Scénario considéré : Prendre le même scénario que l'étude expérimentale 1 en insérant la couche protocolaire de point de reprises implantant l'algorithme de Juang-Venkatesan. Il y a deux phases dans l'exécution de ce scénario :

- le système s'exécute pendant un temps arbitraire sans panne
- un nœud choisi arbitrairement tombe en panne puis supposé redémarrer ce qui déclenche le mécanisme de recouvrement (contrôleur peersim `CrashControler`)

La séquence de ces deux phases se répètent jusqu'au bout de l'expérience afin d'avoir plusieurs échantillons de recouvrements et avoir une meilleure analyse statistique.

Métriques considérées dans cette étude :

- Le temps moyen qu'il faut pour faire un recouvrement
- Le nombre moyen de messages par recouvrement
- Le coût moyen en mémoire du stockage des différents points de reprises pendant une phase d'exécution sans panne

- L'ancienneté moyenne de la ligne de recouvrement, c'est à dire la différence entre l'état du système au moment du démarrage du recouvrement et l'état du système quand le recouvrement se termine. A vous de proposer une méthode de calcul pour estimer au mieux cette métrique.

Paramètres :

- le nombre de nœuds noté N (paramètre du simulateur)
- la complexité en messages de la couche supérieure (paramétrable avec ρ en fonction des résultats de l'étude 1)
- la fréquence de création de point de reprise

Objectifs : Pour le même N constant choisi à l'étude 1, quantifier l'impact de la complexité en messages de la couche supérieure et de la fréquence de création de points de reprise sur les métriques considérées. Le but ultime est de pouvoir trouver, pour une complexité donnée de messages de la couche supérieure, la fréquence de création points de reprise qui offre le meilleur compromis entre coût mémoire et ancienneté de ligne de recouvrement.