SCIENCES SORBONNE UNIVERSITÉ

Projet ARA 2020–2021

Jonathan Lejeune, Célia Mahamdi

Objectif

L'objectif principal de ce projet est de faire une étude expérimentale via le simulateur PeerSim. Il s'agit d'étudier un mécanisme de réplication de machine à états, basé sur le protocole Paxos permettant d'assurer une cohérence sur un ensemble de réplicas d'une même variable logique.

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 pour le 7 février 2021 à 23h59 au plus tard
- 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.

1 Contexte

Un service informatique distant peut être interrogé par des clients via des requêtes. On peut distinguer deux types de requêtes :

- les requêtes dites "de lecture" qui ne modifient pas l'état courant du service.
- les requêtes dites "d'écriture" qui peuvent modifier l'état courant du service

Le déploiement de ce service se fait dans un environnement potentiellement large échelle (beaucoup de clients pouvant amener le service à ne pas pouvoir honorer les requêtes en temps raisonnable suite à une surcharge) et non fiable, aussi bien au niveau des nœuds de calcul (ici le serveur hébergeant le service) qu'au niveau des liens de communication

(les messages entre les clients et le serveur). Ces caractéristiques réalistes peuvent amener à une indisponibilité du service (pas de réponse, forte latence, résultats incohérents, ...). Afin de pouvoir garantir un minimum de disponibilité, il est courant de répliquer ce service. En effet, les requêtes de lecture peuvent requêter directement un réplica ce qui permet une bonne répartition de charge et une transparence des pannes. En revanche, les requêtes d'écriture sont problématiques, car elles remettent en cause la cohérence des états des différents réplicas. L'assurance de cohérence entre les réplicas dans un tel environnement non fiable rend la réplication comme un problème fondamental des systèmes distribués. Pour garder un état cohérent sur les réplicas, il est nécessaire que ces derniers reçoivent les requêtes d'écriture dans le même ordre. On utilise alors un mécanisme de réplication de machine à états qui se base sur un protocole de consensus. Un consensus permet à un ensemble de nœuds de se mettre d'accord sur une valeur de manière durable : une fois que la décision a été prise et validée par une majorité de nœuds, la décision est irrévocable et définitive.

Dans le cadre de ce projet, vous allez implanter et étudier expérimentalement un protocole de réplication de machine à états. Vous vous baserez sur le protocole distribué Paxos conçu par L. Lamport dans les années 90 et qui est un des protocoles les plus utilisés dans les systèmes réels qui font de la réplication.

2 Présentation du modèle

Comme mentionné ci-avant, nous nous plaçons dans un cadre où

1. n'importe quel nœud du système peut tomber en panne n'importe quand
2. une panne peut être définitive ou temporaire
3. un message peut être perdu ou dupliqué
4. les canaux de communications ne sont pas FIFO

Dans notre système, nous considérons un ensemble de nœuds hébergeant les **réplicas** noté $R = r_0, r_1, \ldots$. Chaque nœud réplica r_i est identifié par un indice i tel que $0 \leq i < |R|$.

Chaque réplica $r_i \in R$ maintient un **historique** noté $\mathcal{H}_i$. Cet historique maintient l'ordre des requêtes d'écriture délivrées sur le réplica. On pourra supposer que cet historique sera stocké localement sur un support persistant et fiable ce qui permettra de recouvrir plus rapidement un état cohérent lors d'un redémarrage du nœud. L'indice d'une requête dans un historique est appelé **itération** qui est entier supérieur ou égal à zéro. L'itération x d'un historique $\mathcal{H}_i$ est notée $\mathcal{H}_i(x)$. Ainsi pour tout historique $\mathcal{H}$ du système, si deux itérations x et y existent dans $\mathcal{H}$ et que $x < y$ alors $\mathcal{H}(x)$ désigne une requête qui a été délivrée avant $\mathcal{H}(y)$.

La propriété de **sureté** que doit respecter notre protocole de réplication de machine à états est $\forall \mathcal{H}_i$ et $\forall x \in \mathcal{H}_i$:

- soit $\mathcal{H}_i(x) = \perp$ signifiant que le réplica r_i ne connait pas encore la valeur de la requête associée à l'itération x
- soit $\mathcal{H}_i(x) = req$ tel que $\forall \mathcal{H}_j \neq \mathcal{H}_i \wedge \mathcal{H}_j(x) \neq \perp \Rightarrow \mathcal{H}_j(x) = req$ signifiant que tout autre historique qui maintient une valeur différente de $\perp$ à l'itération x implique que la valeur de cette itération doit être égale à req.

3 Le protocole Paxos

Paxos est un protocole de consensus permettant aux réplicas de se concerter sur une seule valeur. Dans notre contexte une valeur décidée par Paxos serait une seule itération. Il sera donc nécessaire de démarrer une instance de Paxos pour chaque itération de l'ensemble des historiques (cette utilisation de Paxos est aussi appelée Multi-Paxos).

3.1 Principes du protocole

Nous décrivons ici le protocole Paxos basique pour décider d'une valeur (c.-à-d. d'une itération). Les nœuds externes au système sont appelés clients et ce sont ces nœuds qui émettent les requêtes. Parmi l'ensemble des nœuds internes au système, le protocole Paxos définit 3 rôles :

- ***Learner*** **:** les noeuds qui hébergent les réplicas du service
- ***Acceptor*** **:** les noeuds qui permettent de prendre une décision sur la valeur. Ils servent notamment à se rappeler si une valeur a déjà été décidée dans le passé ou bien si une nouvelle valeur doit être décidée. Leur nombre dépend du nombre de fautes que l'on souhaite tolérer. Les `Acceptors` doivent toujours former un quorum pour que le protocole fonctionne. Typiquement un quorum peut être n'importe quel ensemble d'*Acceptor* qui forme une majorité.
- ***Proposer*** **:** les nœuds qui reçoivent les requêtes des clients et qui les soumettent à la décision des Acceptors. Ils sont responsables de récolter la réponse de ces derniers pour valider ou rejeter la requête du client.

Le schéma global du protocole est le suivant (de l'émission du client à l'acceptation définitive de la requête).

Étape 0 Un client choisit un *Proposer* p et lui envoie sa requête.

Étape 1a Le *Proposer* p émet à l'ensemble des *Acceptors* un message `Prepare` contenant un numéro de round (appelé aussi *ballot* ou *numéro de séquence*). Ce numéro de round doit être strictement supérieur à n'importe quel numéro de round déjà envoyé par p. Le but de ce message étant de savoir si un quorum d'*Acceptors* peut être atteint pour pouvoir soumettre la requête du client.

Étape 1b À la réception d'un message `Prepare` sur un *Acceptor* a depuis un *Proposer* p pour un round n, il y a deux cas à considérer :

- ◇ si n est supérieur à n'importe quel round auquel a ait participé, un message `Promise` est envoyé à p. Ce message `Promise` permet d'indiquer à p que a ne participera plus à un scrutin de round inférieur n. Le message `Promise` contient éventuellement la précédente valeur v que a a déjà acceptée (lors d'une précédente phase 2b) associé à son numéro de round. n_v
- ◇ sinon, a ignore le message ou éventuellement renvoi un message `Reject` à p lui indiquant que son numéro de round est invalide et obsolète.

Étape 2a Lorsque p reçoit une majorité de `Promise`, il doit décider d'une valeur e.

- ◇ S' il existe des `Promise` contenant des valeurs déjà acceptées par son envoyeur, alors p choisit la valeur v avec le numéro n_v le plus grand.
- ◇ sinon, p est libre de choisir la valeur et dans notre cas il choisira la requête que le client lui a envoyée à l'étape 0.

Le *Proposer* p envoie alors à l'ensemble des `Acceptors` la valeur e qu'il a choisie associée au numéro de round n qu'il a envoyé dans le message `Prepare` (désigné

parfois aussi comme `Commit`)

Étape 2b À la réception d'un message `Accept` sur un *Acceptor* a depuis un *Proposer* p pour un round n et une valeur e :

- ◇ si n est plus grand ou égal au numéro de round du dernier `Promise` que a ait envoyé, alors il accepte la valeur e. Il mémorise cette valeur (pour les éventuels futurs `Prepare`) et diffuse à l'ensemble des `Learners` qu'à p un message `Accepted` contenant la valeur e.
- ◇ sinon le message est ignoré

Étape 3 Lorsqu'un *Learner* l recoit une majorité de messages `Accepted` pour une même valeur e alors l prend acte de cette décision (la valeur est définitivement acceptée) et exécute la requête décrite par la valeur e. En fonction du protocole applicatif qui caractérise la requête, l peut répondre directement au client.

3.2 Notion de leader

On peut remarquer que plusieurs *Proposers* peuvent rentrer en conflit en phase 1 (étapes 1a et 1b) pour obtenir le numéro de round le plus haut et pouvoir imposer son leadership. De même, il est possible qu'un *Acceptor* change sa valeur précédemment acceptée s'il rencontre un *Proposer* possédant un numéro plus en étape 2b. Cette bataille pour le leadership des Proposer peut amener Paxos à ne jamais converger vers une valeur. C'est pourquoi il est nécessaire que l'ensemble des *Proposers* désigne parmi eux un leader qui assurera une stabilité sur l'exécution du protocole. Cette phase d'élection de leader doit se faire avant la phase 1. Ainsi à la réception d'une requête d'un client sur un proposer p :

- soit p est le leader et commence la phase 1
- soit p n'est pas le leader et peut rejeter la requête du client en indiquant l'identité du leader ou bien retransmettre la requête au leader.

4 Adaptation de Paxos à notre modèle

4.1 Rôle des nœuds

Dans le cadre de ce projet, nous allons faire une simplification très courante qui consiste à considérer que tout nœud interne au système est à la fois `Proposer`, `Acceptor` et `Learner`. Ainsi, si on considère un système à N nœuds où l'ensemble des nœuds est désigné par $\Pi = n_0, n_1, n_2, \ldots, n_{N-1}$ alors :

- tout nœud $n \in \Pi$ a le rôle de *Proposer* autrement dit éligible à l'élection de leader
- tout nœud $n \in \Pi$ a le rôle d'*Acceptor*, ce qui veut dire que la condition pour avoir une majorité est de recevoir $\frac{N}{2} + 1$ message `Promise` et `Accepted` contenant la même valeur pour pouvoir passer en phase 2 et 3
- tout nœud $n \in \Pi$ a le rôle de *Learner* impliquant qu'il y a N réplica du service (donc N historique) dans le système. À la validation d'une nouvelle itération x dans son historique $\mathcal{H}$, avant d'exécuter la requête et répondre au client, un learner devra s'assurer qu'il n'y ait pas d'itération $y < x$ tel que $\mathcal{H}(y) = \perp$. Ceci peut se produire si le noeud est tombé en panne puis a redémarré ou bien si des messages `Accepted` ont été perdus. Si c'est le cas, il faudra relancer un round Paxos (via le leader) pour connaître les valeurs manquantes.

4.2 Élection du leader

Comme nous considérons une réplication de machine à états, il est possible de choisir un identifiant de nœud dans l'itération 0 qui fera office de leader pour les itérations suivantes. On peut ainsi considérer qu'élire un leader, c'est analogue à ce que chaque nœud (en tant que *Proposer*) propose dans l'itération 0 une requête (`findLeader`) émise par eux même (en tant que client). Le leader sera le noeud qui aura été le plus rapide pour valider son identifiant auprès d'une majorité d'*Acceptors*.

Si ce principe peut s'appliquer à l'itération 0, il est également possible de l'appliquer à n'importe quelle itération dès que l'on détecte la panne du leader (voir ci-après). Ainsi il est toujours possible à tout instant qu'un nœud connaisse l'identité de son leader en regardant dans son historique $\mathcal{H}$ (en tant que *Learner*) la dernière requête `findLeader` qui a été validée. Ceci est également applicable lorsqu'un nœud procède au recouvrement de son état à la suite d'une panne.

Afin d'éviter une trop longue convergence (car tous les nœuds en tant que `Proposer` rentrent en conflit pour désigner le leader), il est possible d'introduire un mécanisme de backoff (utilisé notamment dans les détections de collisions sur l'utilisation d'un bus de communication). Ainsi lorsqu'un nœud reçoit un message `Reject` par un *Acceptor* (ce qui signifie que son round n'est pas ou plus valide), il attend pendant une certaine période aléatoire avant de réitérer sa proposition. À chaque rejet, il augmente ce temps d'attente. Ceci permet de réduire à terme les probabilités de conflit et faire en sorte qu'un seul *Proposer* aient le temps de passer les 2 étapes de validation avant qu'un autre *Proposer* redémarre un round plus élevé. À terme, tous les *Proposers* recevront un identifiant de nœud (en tant que `Learner`) qu'ils reconnaîtront comme leader : celui du *Proposer* qui a réussi à passer en premier les deux phases du protocole Paxos.

Il faut bien comprendre que dans une itération `findLeader`, les nœuds (en tant que `Proposer`) ne reconnaissent aucun leader puisque c'est l'objet de la requête. Ils se battent donc pour être le premier à se désigner.

4.3 Surveiller le leader

Afin de contrôler que le leader n'est pas en panne, il est possible d'introduire un mécanisme de pinging. Ainsi, chaque nœud envoie régulièrement un message `Ping` au nœud qu'il croit être leader. À la réception d'un message `Ping` sur un nœud n_i depuis un nœud n_j, n_i renvoie un message `Pong` à n_j avec l'identité du leader de n_i (qui peut être n_i lui-même). À la réception d'un message `Pong` sur n_j, le leader et éventuellement mis à jour.

En cas de non-réception répétée de message `Pong` (détectable via un timeout), le leader peut être considéré comme en panne. Le noeud qui détecte ceci propose à la prochaine itération une nouvelle élection (requête `findLeader`).

Étude expérimentale 1 : itération d'élection

Notre première étude portera uniquement sur l'itération 0 de notre réplication de machine à états, c'est-à-dire l'élection du leader.

Scénario considéré : Tous les nœuds (en tant que *Proposer*) tentent d'élire un leader (via la requête `findLeader`) qui sera utilisé dans les itérations suivantes. Aucune faute ne sera injectée.

Critère de validation : Tous les nœuds doivent avoir la même valeur de leader à la fin du scénario.

Métriques considérées dans cette étude :
- le **temps de convergence** : le temps nécessaire au protocole pour que tous les nœuds reconnaissent le même leader
- le **nombre de rounds** nécessaires pour atteindre une majorité, c'est à dire pour que le consensus soit atteint
- le **nombre de messages** émis

Objectif : Quantifier l'impact des paramètres suivants sur les métriques décrites précédemment :
- le nombre de nœuds dans le système, c'est à dire N
- activation et désactivation d'un temps de backoff. En cas d'activation, on pourra étudier différentes valeurs d'incrémentation qu'un nœud fait à chaque rejet d'un `Prepare`
- la valeur du round initial :
  - ◇ une version avec la valeur 0 (Paxos original)
  - ◇ une version avec l'identifiant du nœud, ce qui donnera normalement plus de chance aux nœuds ayant un plus grand identifiant et donc de converger plus vite.

Étude expérimentale 2 : Multi-Paxos séquentiel

Notre deuxième étude portera sur une exécution du système dans un temps imparti.

Scénario considéré : En dehors d'une itération d'élection, le leader soumet séquentiellement des requêtes. Le leader attend qu'une itération soit validée (en tant que *Learner*) pour passer à la suivante. En fonction des résultats que vous avez eus dans l'étude précédente, vous prendrez la meilleure configuration en ce qui concerne les paramètres de backoff et de round initial dans les itérations `findLeader`.

Critère de validation : La propriété de sûreté décrite à la section 2 doit être vérifiée.

Métriques considérées dans cette étude :
- le **débit moyen de requêtes applicatives validées** qui est le rapport entre le nombre total de requêtes hors élection validées et le temps d'expérience
- la **latence moyenne** qui est le temps moyen pour qu'une requête applicative (hors élection) soit validée
- le **nombre de messages** émis (à l'exception des messages `Ping` et `Pong`)

Objectif : Quantifier l'impact des paramètres suivants sur les métriques décrites précédemment :

- le nombre de nœuds dans le système, c'est à dire N
- probabilité qu'un nœud tombe en panne
- probabilité pour qu'une panne soit franche ou transitoire

On s'assurera que le nombre de nœuds en panne franche soit toujours en minorité afin de garantir que le protocole fonctionne

Étude expérimentale 3 : Réduire le coût en messages

Vous avez pu remarquer précédemment qu'un consensus pouvait être très coûteux en messages. Ceci est une source de saturation de la bande passante du réseau. Dans la pratique, un message est en effet composé d'entêtes des différentes couches réseau (TCP/UDP, IP ...) et d'un contenu applicatif. Dans la plupart des cas, la taille du contenu des messages est petite par rapport à la taille des en-têtes. Par conséquent, une forte complexité en messages implique une bande passante surtout utilisée pour les données des en-têtes.

Pour réduire ce coût en bande passante, il existe deux leviers d'actions :

- réduire le nombre de message dans une des phases du protocole Paxos
- produire des messages avec des tailles de contenus applicatifs plus grandes en faisant par exemple de la bufferisation (un seul envoi concerne en réalité plusieurs messages)

Objectifs : Proposez une ou plusieurs solutions pour réduire le coût en messages. Vous évaluerez votre ou vos solution(s) à travers une étude expérimentale. Dans votre rapport vous devrez :

- parler des éventuelles modifications que vous avez apportées au modèle (exemple : ajout d'hypothèses)
- décrire votre ou vos solution(s)
- décrire votre protocole expérimental de votre étude
- analyser vos résultats en mettant en évidence :
  - ◇ les gains de votre solution.
  - ◇ éventuellement les contreparties de votre solution (ex : dégradation d'une autre métrique, hypothèse plus forte ...)
- synthétiser vos résultats