SCIENCES SORBONNE UNIVERSITÉ

AR (4I403) - PROJET : Implémentation du protocole P2P CHORD

CHORD est une table de hachage distribuée (DHT). Cela signifie que l'objectif du système est de stocker des données de manière distribuée et d'associer une clé à chacune d'elle. De plus, le système doit fournir des fonctions permettant de retrouver une donnée à partir de sa clé, de stocker une donnée (en construisant sa clé), de supprimer une donnée du système, etc. Enfin, vu le contexte des réseaux P2P qui sont par essence hautement dynamiques, le protocole doit intégrer une gestion du départ et de l'arrivée de nœuds dans le système (celle-ci peut être volontaire ou subie). Pour plus de détails, se référer à : I. Stoica, R. Morris, D. Karger, M. Kaashoek, H. Balakrishnan : *Chord : A scalable peer-to-peer lookup service for internet applications*, SIGCOMM 2001, pp. 149-160.

Le but de ce projet est d'implémenter à l'aide de la bibliothèque MPI des fonctionnalités simplifiées de la DHT CHORD. Ce projet comporte trois exercices indépendants et à traiter dans des fichiers séparés dont les objectifs respectifs sont :

1. Implémenter la recherche d'une clé dans une DHT CHORD bénéficiant d'un calcul centralisé des finger tables.
2. Proposer un algorithme de calcul distribué (simplifié) des finger tables puis l'implémenter.
3. Proposer un algorithme d'insertion d'un nouveau pair dans une DHT CHORD existante et, de manière optionnelle, l'implémenter.

Votre compte-rendu sera constitué uniquement des trois fichiers relatifs à ces trois exercices (on donne plus de détails sur leur contenu dans l'énoncé). Celui-ci devra être soumis **via Moodle** au plus tard le **10 mai 2020 à minuit** et comporter de manière claire les noms des deux membres du binôme le cas échéant.

Dans ce projet, on s'intéresse à une version simplifiée de CHORD dont voici les caractéristiques. L'ensemble des pairs du système est noté Π. Les identifiants des pairs sont prises dans un ensemble I. L'ensemble des données est noté D. Les identifiants des données (*i.e.* les clés) sont prises dans un ensemble K. On suppose que $I = K$ est l'ensemble des entiers $\{0, \ldots, 2^M - 1\}$ (avec M une constante de votre programme) donc toute clé est encodée sur M bits.

On dispose de deux fonctions de hachage $f : \Pi \longrightarrow I$ et $g : D \longrightarrow K$. Pour simplifier, on considère que f et g sont des fonctions aléatoires (garantissant l'unicité des valeurs tirées pour chaque ensemble).

L'ensemble Π des pairs est arrangé en anneau dans l'ordre croissant de leur identifiant (dans le sens des aiguilles d'une montre). Chaque pair de rang MPI p dispose des variables :

— $id_p = f(p)$ son identifiant CHORD ;
— $finger_p$ un tableau contenant ses M fingers comme vu en cours et TD (il est nécessaire de stocker pour chaque finger son identifiant CHORD mais aussi son rang MPI pour pouvoir lui envoyer des messages) ; et
— $succ_p = finger_p[0]$ l'identifiant CHORD de son successeur dans l'anneau.

Chaque pair p est *responsable* des données d'identifiant CHORD dans l'intervalle $]id_q; id_p]$ où q est le pair tel que $succ_q = p$ (notez que q est inconnu du pair p), c'est-à-dire qu'il doit stocker ces données et répondre aux requêtes portant sur ces données.

Remarque importante : Il ne vous est pas demandé de gérer le stockage des données elles-mêmes mais simplement de gérer la responsabilité des clés.

Exercice 1 – Recherche d'une clé (6 points)

Dans cet exercice, un processus simulateur initialisera la DHT CHORD **de manière centralisée** : il calculera l'ensemble des finger tables après avoir tiré aléatoirement les identifiants des pairs.

Question 1 : Implémentez l'initialisation de la DHT par le processus simulateur. Inspirez-vous des TME précédents.

Question 2 : Implémentez la recherche du pair responsable d'une clé CHORD par un pair quelconque avec l'algorithme vu en TD.

Question 3 : Testez votre implémentation de la manière suivante : après avoir initialisé le système, le processus simulateur tire de manière aléatoire un identifiant de pair (pensez à vérifier son existence) et une clé de donnée puis demande (à l'aide d'un message spécial) à ce pair de chercher le responsable de cette donnée. Pensez à réaliser des affichages pour vérifier le bon déroulement de la recherche en fonction de l'état de la DHT. Une fois la requête terminée, le pair en informe le processus simulateur qui propage alors un message de terminaison à tous les processus.

Compte-rendu : Votre code **convenablement commenté**.

Exercice 2 – Calcul des finger tables (9 points)

Dans cet exercice, nous souhaitons réaliser l'initialisation de la DHT CHORD **avec une complexité en messages sous-quadratique** (*i.e.*, inférieure à $|\Pi|^2$) **de manière distribuée** (*i.e.,* sans faire intervenir le simulateur). Pour cela, nous supposons que les pairs sont initialement organisés en anneau bidirectionnel (pas nécessairement ordonné en fonction des identifiants des pairs) et qu'un nombre **non nul quelconque** de pairs peuvent être initiateurs du calcul des finger tables.

Question 1 : Proposez un algorithme permettant de réaliser un tel calcul des finger tables. Il vous est conseillé d'utiliser des algorithmes connus.

Question 2 : Implémentez votre algorithme. Le processus simulateur se contentera de tirer aléatoirement les identifiants CHORD des pairs puis de construire un anneau dans lequel les rang MPI seront ordonnés et de déterminer aléatoirement un ensemble non vide d'initiateurs parmi ces pairs. Une fois la construction des finger tables terminée, chaque pair affiche la sienne puis se termine.

Compte-rendu : La description **précise** de votre algorithme, la justification de sa correction, la justification de sa complexité et votre code **convenablement commenté**.

Exercice 3 – Insertion d'un pair (5 points + bonus)

Dans cet exercice, nous supposons avoir une DHT CHORD correctement initialisée. Nous supposons de plus que tout pair de rang MPI p dispose d'une liste $inverse_p$ contenant l'identifiant (et le rang MPI) de tout pair q ayant un finger sur p (*i.e.*, il existe un k tel que $finger_q[k] = id_p$).

Nous souhaitons réaliser l'insertion d'un nouveau pair dans cette DHT **avec une complexité en messages moyenne sous-linéaire** (*i.e.*, inférieure à $|\Pi|$) **de manière distribuée** (*i.e.,* sans faire intervenir le simulateur). Cette insertion doit garantir que, une fois terminée, la DHT est dans un état global cohérent avec sa spécification (en particulier, les finger tables de tous les pairs doivent être cohérentes avec le nouvel état de la DHT).

On supposera que le pair entrant connaît initialement son identifiant (avec la garantie que cet identifiant n'est pas présent dans la DHT). De plus, il ne sera capable initialement de n'envoyer des messages qu'à un unique pair (choisi arbitrairement) de la DHT. Pour envoyer un message à d'autres pairs, il devra être informé de leur existence par un pair déjà présent dans la DHT.

Question 1 : Proposez un algorithme permettant de réaliser une telle insertion.

Question 2 (BONUS) : Implémentez votre algorithme. Pour l'initialisation de la DHT, vous pouvez reprendre celle de l'exercice 1 en la modifiant de façon à construire les listes $inverse$. Une fois la DHT initialisée, le processus simulateur signalera à un processus (qu'il n'aura pas inclus dans la DHT) qu'il peut déclencher son insertion en lui envoyant son identifiant (en prenant garde à l'unicité de cet identifiant) et celui d'un pair de la DHT (choisi aléatoirement). Une fois l'insertion terminée, le pair en informe le processus simulateur qui propage alors un message de terminaison à tous les processus (qui afficheront leur état pour vérifier si l'insertion s'est correctement déroulée).

Compte-rendu : La description **précise** de votre algorithme, la justification de sa correction, la justification de sa complexité et (de manière optionnelle) votre code **convenablement commenté**.