UPMC
SORBONNE UNIVERSITÉS

# 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 noeuds dans le système (celle-ci peut être volontaire ou subie). Pour plus de détails, se référer à : Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, Hari 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 deux versions simplifiées de la DHT CHORD. La première simplifiera le routage utilisé mais vous demandera d'implémenter la recherche d'une donnée ainsi que la suppression et l'ajout d'un pair au système. La seconde utilisera le vrai routage de CHORD mais ne vous demandera d'implémenter que la recherche d'une donnée.

Votre compte-rendu sera constitué du code source de chacune de ces deux versions convenablement commentés. Celui-ci devra être envoyé par mail (swan.dubois@lip6.fr) au plus tard le **23 avril 2017 à minuit** et comporter de manière claire les noms des deux membres du binôme le cas échéant.

## Exercice 1 – CHORD avec routage simplifié

Dans cet exercice, on s'intéresse à une version simplifiée de CHORD dont voici les caractéristiques. L'ensemble des pairs du système est noté $\Pi$. 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 exister $k \in \mathbb{N}$ tel que $I = K = \mathbb{Z}/k\mathbb{Z}$.

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 $\Pi$ des pairs est arrangé en anneau dans l'ordre croissant de leur identifiant (dans le sens des aiguilles d'une montre). Chaque pair $p$ connait :

— $id_p = f(p)$ son identifiant CHORD,
— $succ_p$ l'identifiant CHORD de son successeur dans l'anneau et
— $resp_p = id_q + 1$ avec $succ_q = p$.

Chaque pair $p$ est *responsable* des données d'identifiant CHORD dans l'intervalle $[resp_p; id_p]$, c'est-à-dire qu'il doit stocker ces données et répondre aux requêtes portant sur ces données.

Noter que l'on considère ici un routage simplifié par rapport au protocole CHORD : chaque pair ne peut communiquer qu'avec son successeur dans l'anneau logique (pas de *finger table*). Cela conduira donc à de moins bonnes performances du protocole (par exemple, la recherche d'une donnée sera linéaire par rapport au nombre de pairs au lieu d'être logarithmique).

## Étape 1 : Initialisation du Système

Le programme devra être réalisé sous MPI. Un processus simulateur va créer une topologie de bootstrap composée de $n$ pairs en anneau. En début de programme, le processus simulateur enverra à chaque pair du système :

— sa clé CHORD choisie aléatoirement dans l'ensemble des identifiants CHORD sur $m$ bits où $m$ est une constante du protocole,
— la clé CHORD de la première donnée dont le pair est responsable et
— la clé CHORD du successeur du pair dans l'anneau.

Remarque : Pour des raisons dues à l'identification des processus sous MPI, vous devez également stocker le rang MPI correspondant au successeur (pas le prédécesseur car on ne communique jamais directement avec lui).

La suite de l'exercice consiste à faire gérer par le système un certain nombre d'événements dans un ordre prédéfini (on ne vous demande pas ici de faire un système "opérationnel" qui gère des évènements dans un ordre inconnu à l'avance). Il ne vous est pas demandé de gérer le stockage des données elles-mêmes mais simplement de gérer la responsabilité des clés.

## Étape 2 : Recherche d'une Donnée

Une fois la phase d'initialisation (étape 1) finie, un des pairs va lancer une requête de recherche. Le but est d'écrire un protocole permettant à ce pair de connaître la clé CHORD du pair responsable d'une donnée identifiée par sa clé CHORD (la clé de cette donnée peut par exemple être une variable globale de votre programme que vous choisissez arbitrairement).

## Étape 3 : Gestion de la Dynamicité du Système

L'objectif de cette étape est de gérer des ajouts et suppressions de pairs dans le système. Pour simplifier le problème, nous supposons que tout pair notifie le système avant de se déconnecter du système (pas de panne brutale).

Dans un premier temps, un des pairs du système va se déconnecter. Le système doit alors retrouver un état cohérent grâce à la notification. Dans un second temps, ce pair va se reconnecter (il peut utiliser une clé CHORD différente si vous le souhaitez). Le système doit alors redistribuer la responsabilité de certaines données afin de respecter la spécification du système.

Remarque : Le pair qui se connecte commence par contacter un processus choisi arbitrairement afin de connaître la position à laquelle il doit s'insérer dans l'anneau.

# Exercice 2 – CHORD sans routage simplifié

On reprend le système CHORD de l'exercice précédent mais on y adjoint la gestion des finger tables vues en cours et en TD.

## Étape 1 : Initialisation du Système

Le programme devra être réalisé sous MPI. Un processus simulateur va créer une topologie de bootstrap composée de $n$ pairs en anneau. En début de programme, le processus simulateur enverra à chaque pair du système :

— sa clé CHORD choisie aléatoirement dans l'ensemble des identifiants CHORD sur $m$ bits où $m$ est une constante du protocole,
— la clé CHORD de la première donnée dont le pair est responsable et
— la clé CHORD et le rang MPI de chaque pair figurant dans la finger table du pair.

Remarque : Les finger tables seront calculées de manière centralisée par le processus simulateur *avant* l'initialisation du système.

## Étape 2 : Recherche d'une Donnée

Une fois la phase d'initialisation (étape 1) finie, un des pairs va lancer une requête de recherche. Le but est d'implémenter l'algorithme vu en TD permettant à ce pair de connaître la clé CHORD du pair responsable d'une donnée identifiée par sa clé CHORD (la clé de cette donnée peut par exemple être une variable globale de votre programme que vous choisissez arbitrairement).

## Étape Bonus

Implémentez le calcul réparti des finger tables (celles-ci ne seront plus diffusées initialement par le processus simulateur mais calculées de manière réparti par l'ensemble des pairs). Vous prendrez garde à minimiser la complexité de votre solution.