INFO833 - Systèmes distribués à large échelle

Compte Rendu TP1 – DHT

SOUCHON Romain

PASCAL Quentin

GitHub : https://github.com/QuentinP-PEIP/INFO833_DHT

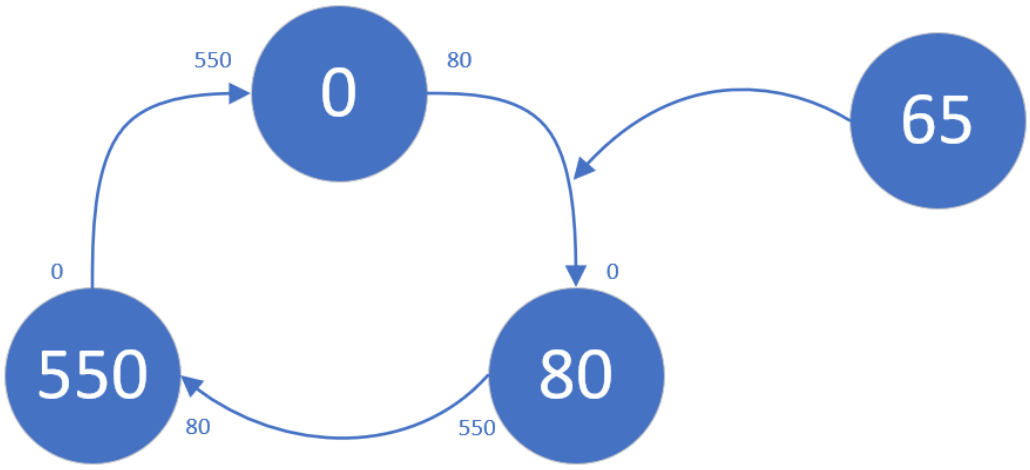
Initialiser notre DHT :

On commence par attribuer un id unique à chaque nœud, un uid. Après avoir créé tous les nœuds, on initialise les nœuds voisins pour les trois premiers nœuds du réseau. Le but est de commencer avec trois nœuds qui envoient un message à leur nœud voisin suivant.

Partie 1 – Join/Leave :

Fonctionnement :

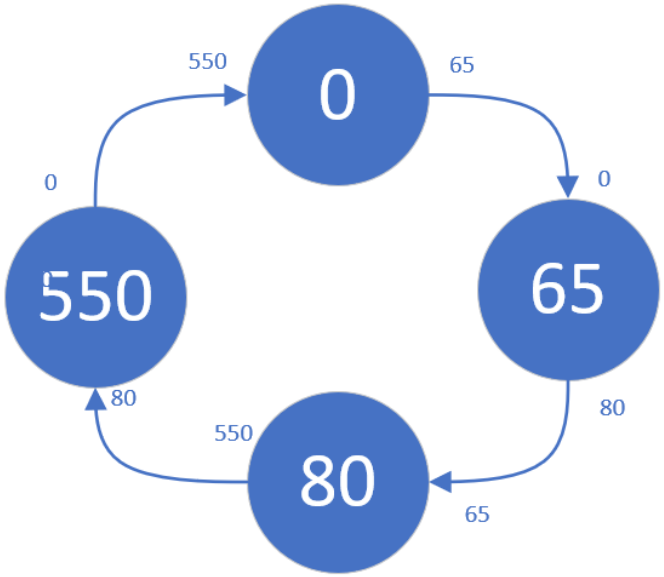
1) On ajoute un événement avec un Message de type JOIN
2) On utilise plusieurs constructeurs de Message pour transporter certaines informations (nœud ou uid), dans JOIN, on envoi un message de type PLACE au nœud 0 afin de trouver la place du nœud à ajouter.



3) On parcourt la DHT dans le sens croissant des uid. On compare l'uid du nœud à placer avec le nœud suivant.
    a. Si l'uid du nœud à placer est inférieur à l'uid du nœud suiv, on set les nœud du nouveau nœud et on envoi des messages du type SET_NOEUD_PREC et SET_NOEUD_SUIV.

b. Si l'uid du nœud suivant est le 0, alors le nœud à ajouter à la plus grande uid de la DHT, on le place
c. Si l'uid du nœud à placer est inférieur à l'uid du nœud suiv, on envoyer un message PLACE au nœud suivant

4) Les messages SET_NOEUD_PREC et SET_NOEUD_SUIV ont pour but d'annoncer aux nœuds voisins du nœud ajouté, son arrivé et donc de set ses nouveaux voisins.



On remarque que si on envoie un message au nœud 0, le nœud a bien été ajouté à la DHT.

Pour le message LEAVE, on utilise les messages SET_NOEUD_PREC et SET_NOEUD_SUIV, en envoyant le nœud précédent au nœud suivant et inversement. Pour finir, on enlève les voisins du nœud enlevé.

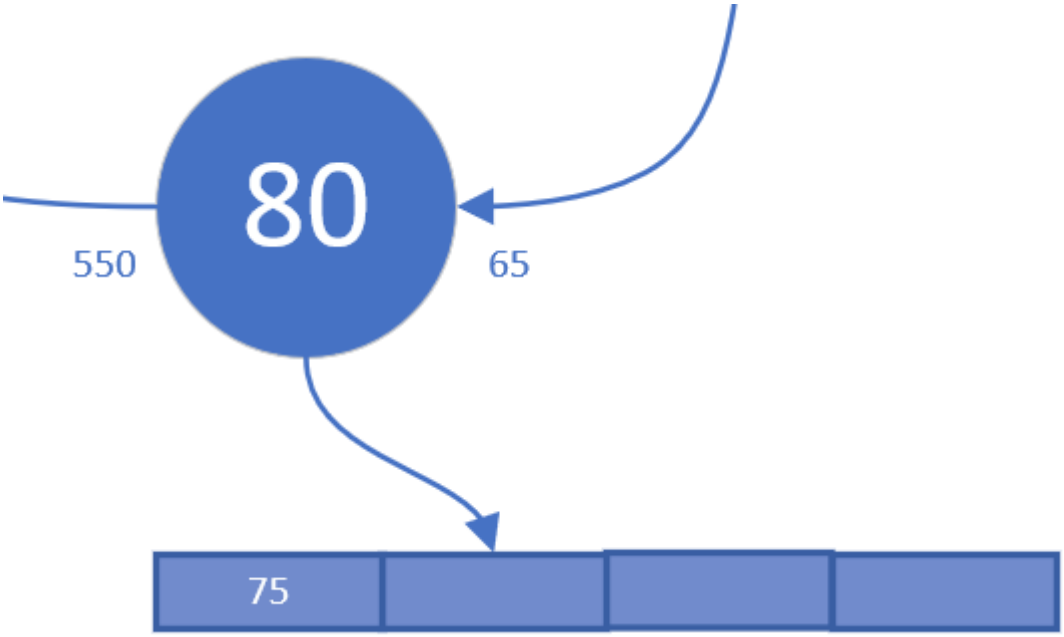
Partie 2 – Send/Deliver :

Afin de savoir si un nœud n'est pas tombé en panne, nous allons faire en sorte que chaque nœud a à envoyer un ping au nœud 0 (ou à un autre nœud) pour avertir qu'il est toujours en activité. Nous allons donc créer des messages du type « SEND ». Une fois reçu, le nœud envoi donc un ping par l'intermédiaire d'un message de type « DELIVER ». Pour vérifier l'activité de tous les nœuds, il faut garder en historiques quels nœuds ont envoyé leur message et lesquels ne l'ont pas fait.

Partie 3 – Put/Get :

Nous créé une classe Data dans laquelle chaque objet est identifié par un id. Chaque nœud possède une liste de données, au début vide mais dans laquelle se stockeront les données ajoutées. Quand un message de type « PUT » est envoyé à

un nœud avec une donnée à stocker, le but est de trouver dans quel nœud la stocker. Pour se faire, nous commençons par envoyer un message de type « PLACE_DATA » au nœud 0, puis nous allons comparer les valeurs absolues de la différence entre l'id de la donnée et l'id du nœud actuel avec celle de l'id de la donnée et l'id du nœud suivant. Si la première différence est inférieure ou égale à la seconde, alors on attribut la donnée à ce nœud. De plus pour afin de sécuriser la donnée, on la stocke aussi sur le nœud suivant et le nœud précédent. Sinon, on envoie un message de type « PLACE_DATA » au nœud suivant.



Une fois la donnée stockée, on peut envoyer un message de type « GET » qui va parcourir la DHT pour trouver quel nœud possède la donnée recherchée.

Partie 4 – Advanced Routing :

Ne sachant pas comment améliorer le transit des messages en trichant, nous avons opté pour l'utilisation d'une table de routage stockée dans le nœud 0. Notre table de routage est une ArrayList contenant les différents nœuds de la DHT triés suivant leur uid. Il suffira juste de parcourir la liste afin de savoir où se placer en rejoignant la DHT ou alors où stocker/récupérer une donnée sans avoir une approche de proche en proche.

Quand un nœud reçoit un message, on envoi un message au nœud 0 et s'il n'est pas dans la table de routage, il est ajouté. De même quand on ajoute un nouveau nœud dans la DHT. Lorsqu'un nœud quitte la DHT, il est enlevé de la table de routage.