**Rapport de Projet : Implémentation d'une DHT avec SimPy**

**Introduction**

L'objectif de ce projet est de concevoir et implémenter une table de hachage distribuée (DHT) en utilisant SimPy. Une DHT permet un stockage et une récupération efficace des données dans un réseau pair-à-pair. Le projet a été divisé en plusieurs étapes :

1. Gestion de l'ajout et du retrait des nœuds (Structure de la DHT)
2. Routage des messages.
3. Stockage, réplication récupération des données

Nous avons choisi d’utiliser SimPy pour sa flexibilité d’utilisation et pour les nombreuses librairies que proposent python.

**1. Structure de la DHT**

Dans cette première étape, chaque nœud a un identifiant unique et est organisé dans un anneau basé sur cet identifiant. Chaque nœud ne connaît que ses voisins immédiats.

Ajout d'un nœud

Lorsqu'un nouveau nœud rejoint la DHT, il contacte le nœud d’origine. Il trouve sa position dans l'anneau en envoyant des messages aux nœuds voisins jusqu'à ce qu'il détermine son emplacement optimal. Une fois inséré, il informe ses nouveaux voisins qui mettent à jour leur voisins.

Suppression d'un nœud

Lorsqu'un nœud quitte la DHT, il informe ses voisins gauche et droit, qui se reconnectent directement entre eux pour maintenir la structure de l'anneau. Si le nœud d’origine quitte on utilise alors un processus aléatoire qui déterminera si le nouveau nœud d’origine est son voisin droit ou gauche.

Dans une perspective d’améliorations il faudrait que le choix du futur nœud d’origine ne soit pas prévisible pour des raisons de sécurité. Si un attaquant peut prédire le futur nœud d’origine et qu’il arrive a ce placer au bon endroit il pourra donc prendre le contrôle et empêcher l’arrivée de nouveau nœud.

**2. Routage des Messages**

Une fois que l'anneau est fonctionnel (plus de 3 nœuds), les nœuds peuvent s'envoyer des messages. Chaque message contient un expéditeur, un destinataire et un contenu.

Fonctionnement

Lorsqu'un nœud souhaite envoyer un message à un autre, il vérifie si le destinataire est l'un de ses voisins. Si ce n'est pas le cas, il transmet le message à son voisin le plus proche du destinataire, optimisant ainsi le routage. Si l’identifiant du destinataire est supérieur a celui de son voisin de droite il transmet le message au voisin de droite et réciproquement.

**3. Stockage et Réplication des Données**

La dernière étape consiste à permettre aux nœuds de stocker des données.

Attribution de l’ID des données

Chaque donnée a un identifiant unique, cet identifiant est choisi aléatoirement parmi un nombre intervalle qui correspond aux nombres de nœuds maximum. Cette solution est assez basique puisqu’elle peut poser des problèmes de répartition. Dans une perspective d’améliorations il faudrait appliquer un double modulo sur l’identifiant. Le premier modulo avec le nombre max de nœuds et le deuxième avec le nombre maximum de données. Cela permettrait une répartition plus équilibrée des données dans la DHT.

Insertion et Réplication des données

La donnée s’insère dans la DHT sur le même principe que l’insertion d’un nœud. Le nœud qui veut stocker une donnée l’envoie à son voisin de droite ou de gauche et elle se déplace ensuite par échange de message entre les nœuds en étant le contenu du message. Une fois qu’elle a trouvé le nœud sur lequel être stocké, le nœud devient le propriétaire de la donnée et l’envoie à ses voisins proches afin que la donnée soit répliquée (degré de réplication = 3).

Une image contenant dessin, croquis, cercle, Dessin au trait

Le contenu généré par l’IA peut être incorrect.Lorsqu’un nœud est ajouté à la DHT, les voisins du nouveau nœud mettent à jour les données. Ce processus consiste à renvoyer les données qui ont besoin d’être répliquée au nouveau nœud et à supprimer les données des nœuds qui ne sont plus voisines direct du propriétaire.

Une image contenant croquis, dessin, cercle, Dessin au trait

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, cercle, diagramme, capture d’écran

Le contenu généré par l’IA peut être incorrect.Ici quand le nœud 8 s’insère, 2 le propriétaire de la donnée 5 lui envoie et le nœud 13 supprime la donnée de ses données car son propriétaire n’est plus son voisin proche. Et même fonctionnement pour la donnée 16.

**Difficultés rencontrées**

Nous avons initialement conçu la DHT sans vraiment échanger des messages. Nous utilisions les propriétés du langage objet de python pour mettre à jour directement les attributs des nœuds voisins sans réellement communiquer avec eux. Afin d’avoir une simulation plus réaliste nous avons donc créer pour chaque nœud une « boite de réception » qui fonctionne en permanence. Chaque nœud qui veut communiquer avec son voisin doit alors lui envoyer un message (que ce soient des messages « classiques » ou des messages permettant au nœud de trouver sa place). Nous avons donc dû réadapter notre code avec l’utilisation des messages ce qui a pris du temps et nous a forcé à repenser notre manière de stocker et répliquer les données.

La mise à jour du stockage des nœuds a également posé des problèmes. Etant dans un simulateur a évènements discrets les évènements prennent un certain temps à se réaliser. Or dans nos fonctions python nous avons commis l’erreur de penser qu’une commande écrite avant une autre serait forcément exécutée avant. Cela a donc poser problème lorsqu’un nœud s’insérait. En effet nous lancions le processus de mise a jour du stockage des données alors que les nœuds n'avaient pas fini de communiquer entre eux afin que chacun connaisse sa place ce qui a engendré un mauvais emplacement pour les données et leurs reliquats.

**Perspectives et Améliorations**

Optimisation du routage

Le routage actuel de proche en proche n’est pas très performant. A grand échelle envoyer un message à travers l’entièreté de la DHT prendra trop de temps et de ressources. Avec plus de temps nous aurions essayé d’implémenter un système de liens longs ce qui auraient permis que certains nœuds connaissent des données plus loin d’eux et ainsi transmettre plus rapidement les informations

Tolérance aux pannes

Actuellement le système n’est pas tolérant aux pannes. Ainsi si un nœud meurt pendant un processus d’envoie de messages le message tournera indéfiniment. Avec plus de temps nous aurions aimé implémenter un système de ping ou un nœud envoie toutes les 10 secondes un message à ses voisins pour vérifier s’ils sont toujours connectés.

**Conclusion**

Ce projet a permis de concevoir et d’implémenter une table de hachage distribuée (DHT) fonctionnelle en utilisant SimPy. Nous avons développé les mécanismes essentiels, notamment la gestion dynamique des nœuds, le routage des messages et le stockage des données avec réplication. Grâce à cette approche, nous avons pu mieux comprendre les défis liés aux systèmes distribués et aux simulations basées sur des événements discrets.

Toutefois, certaines limitations restent à corriger. L’optimisation du routage permettrait d’améliorer l’efficacité du système à grande échelle, tandis que l’introduction d’une tolérance aux pannes renforcerait la robustesse du réseau. Enfin, une meilleure répartition des données et une gestion plus fine des départs de nœuds pourraient compléter notre implémentation. Ce travail nous a permis d’acquérir des compétences en simulation, en conception de systèmes distribués et en gestion des ressources dans un environnement asynchrone.