TD/TP mise en place d'un système réparti et introduction à l'algorithmique répartie

Il est question de revoir les notions liées à la programmation d'applications client-serveur en utilisant les sockets et de se familiariser avec une conception répartie d'un algorithme distribué.

Vous ne souhaitez pas passer à coté d'une approche correcte du travail que vous allez fournir et souhaitez être efficace ? Lisez bien les sujets : chaque détail a son importance.

1 Exemple introductif : quel est le nombre de sites dans mon réseau ? (durée approximative 3h)

L'objectif de cet exercice est de mettre en place un réseau sous forme d'un anneau orienté (graphe dirigé) et de calculer (de manière répartie) la taille de cet anneau.

Prenons un exemple : nous souhaitons mettre en place un anneau impliquant 4 processus P_1, P_2, P_3 et P_4. Les envois de messages dans cet anneau sont possibles dans un seul sens : P_1 peut envoyer des messages à P_2, P_2 à P_3, P_3 à P_4, et P_4 à P_1. Chaque processus P_i doit répondre à la question : quel est le nombre de site dans l'anneau ? Attention, ce n'est qu'un exemple.

Deux étapes sont nécessaires :

1. Créer un anneau : une fois les processus lancés sur des machines différentes, il est question de créer les liens entre chaque processus et ses voisins. Les liens entre voisins seront matérialisés à l'aide de sockets utilisant le protocole de transport TCP.
2. Définir et implémenter un algorithme réparti qui a pour but que chaque site calcule la taille de l'anneau (résultat final pour l'exemple est de 4 et encore une fois ce n'est qu'un exemple).

Voici quelques indications :

1. Lorsqu'un processus P_i est lancé, il obtient en paramètre son indice i et il n'a aucune information préalable sur les autres processus P_j ni sur la taille de l'anneau.
2. Pour construire le graphe d'interconnexion, une solution est d'avoir un processus supplémentaire, qu'on appellera P_{config}. Chaque processus P_i informera P_{config} de son existence (l'adresse de P_{config} sera à passer en paramètre de P_i). Il communiquera aussi à P_{config} ce qui est nécessaire pour que ce dernier puisse le mettre en relation avec ses voisins.
3. Une fois que P_{config} aura "construit" l'anneau, il termine (il n'a aucun rôle à jouer dans l'algorithme réparti à mettre en oeuvre).
4. La construction du graphe n'a aucun besoin de programmation concurrente.
5. Un noeud P_i doit communiquer avec un voisin P_j en utilisant TCP. Un seul canal de communication sera à créer (éviter les connexions multiples inutiles entre deux processus et qui seraient considérées comme une erreur). Entre les processus P_i et P_{config}, les échanges peuvent se faire en UDP ou en TCP. Faites simple et ayez juste un moyen de s'assurer que chaque processus P_i est bien connecté à ses voisins.
6. Une fois un processus P_i est connecté à tous ses voisins, il peut commencer son calcul. Il n'y a aucun besoin que le réseau en entier soit construit avant (rappel : dans un algorithme réparti, un processus a une vision locale limitée à son voisinage).
7. Proposer un algorithme pour P_i impliquera la définition d'un protocole d'échange (envoi/réception de quoi et vers/depuis qui). Il en est de même pour les échanges entre P_i et P_{config}.
8. Pour utiliser des machines différentes, vous avez la possibilité d'utiliser, via *ssh*, les machines prodpeda-x2go1, prodpeda-x2go2, ..., prodpeda-x2go6. Vous avez aussi la possibilité de travailler en petit groupe (2 ou 3 membres) et d'utiliser en plus vos postes de travail pour construire un réseau. Le but est d'utiliser un nombre de machines $1 < M \leq nb\ de\ processus$ P_i. D'autres possibilités s'offrent à vous en utilisant par exemple le réseau internet en étant hors réseau local de la FdS.

2 Création d'un graphe d'interconnexion quelconque (durée approximative 4h30

L'objectif de cet exercice est de développer un processus P_{config} (voir précédent exercice) permettant d'interconnecter un ensemble de processus suivant n'importe quel graphe d'interconnexion. L'idée est de pouvoir réutiliser ce programme dans l'ensemble des TP suivants et impliquant l'implémentation d'un algorithme distribué. En réussissant cette étape, vous allez pouvoir vous consacrez pleinement à la partie algorithmique distribuée et optimiserez votre temps de travail.

Pour la réalisation de cet exercice, l'idée est de s'appuyer sur ce qui se fait couramment et qui consiste à décrire un graphe d'interconnexion dans un fichier texte. Ce fichier sera donnée en paramètre du processus P_{config} et sera analysé par ce dernier pour déduire le nombre de processus et le voisinage de chaque processus.

Pour ce faire, il est question d'utiliser le format de fichier proposé sur le site suivant (colonne "Graph" : `http://cedric.cnam.fr/~porumbed/graphs/`. Ce site traite de l'algorithmique de coloration de graphe mais ceci n'est pas le sujet de ce TP.

1. Etudier pour bien comprendre le contenu de la description d'un graphe
2. Définir une stratégie de mise en relation des processus P_i, tout en respectant les indications de l'exercice précédent (utilisation de TCP entre les processus P_i, UDP ou TCP entre P_{config} et P_i, un seul canal de communication entre deux processus voisins, etc.)
3. Construire quelques exemples de graphes de petite taille.
4. Développer un programme pour P_{config} et un programme pour P_i. Le programme de P_i dans le cadre de cet exercice ne contiendra que la partie interconnexion avec ses voisins. En remplacement de la partie algorithmique qui est sensée suivre, faire simplement une demande de saisie au clavier.

Quelques indications

1. Commencer l'implémentation par la lecture des fichiers formatés pour extraire les données utiles, ceci avant de passer à la suite.
2. Prendre le temps de trouver une stratégie de manipulation des données du fichier texte en entrée afin d'aboutir à une solution réduisant l'espace mémoire à utiliser et simplifiant la mise en place des interconnexions sans aboutir à des situations d'interblocage.
3. Pour pouvoir tester votre code sur des graphes de tailles variables et le passage à l'échelle, développer une solution pour (semi)-automatiser le lancement des processus P_i (toujours en ciblant une exécution sur plusieurs machines).