INFO4B Sujet 1 : Calcul distribué de la persistance des nombres

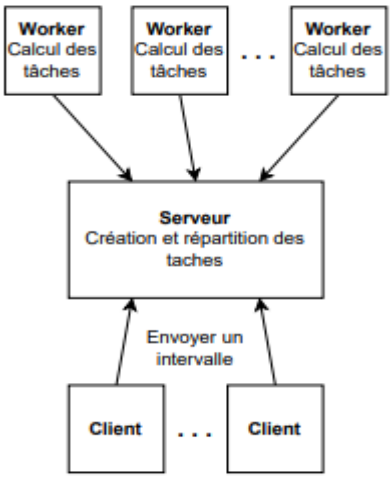
GUIBARD Théo
GIE Jimmy
L2 IE

Introduction :

Objectifs du projet :

L'objectif principal du projet est de développer un système distribué pour le calcul de la persistance multiplicative des nombres. Nous devons exploiter la puissance de plusieurs machines pour répartir le traitement d'un grand nombre de tâches et ainsi bénéficier d'une grande capacité de calcul. Le système sera composé d'un serveur chargé de produire les tâches à exécuter et de les distribuer à des workers, qui sont des machines distantes. Les workers vont exécuter les calculs puis renvoyer les résultats au serveur. L'idée est de pouvoir explorer un grand nombre d'indices pour obtenir la persistance multiplicative de chacun d'entre eux.

Le serveur doit être capable de gérer les connexions des workers et de répartir les tâches équitablement en fonction de leur charge de travail actuelle. Le serveur doit également pouvoir stocker les résultats et les enregistrer périodiquement sur le disque. Enfin, un programme client doit permettre de superviser le système en affichant des statistiques, en demandant les résultats sur un intervalle spécifique, ou encore en consultant la persistance d'un nombre.



Les tâches à réaliser :

Plusieurs machines de travail à distance exécuteront des tâches pour ce projet, qui nécessite un serveur pour superviser la production et la distribution des tâches. Le serveur doit suivre attentivement la charge de travail de chaque travailleur et répartir les tâches en tenant compte de la disponibilité de chaque machine. Les résultats seront renvoyés au serveur pour gestion. Pour exécuter ses tâches, le serveur attribuera des intervalles aux travailleurs qui testeront ensuite les nombres en expansion. Les résultats stockés seront régulièrement enregistrés sur le disque pendant que le serveur reste actif tout au long. Les travailleurs doivent signaler leur disponibilité et le nombre de cœurs disponibles lors de la connexion au serveur.

La configuration comprend un programme client de surveillance pour la configuration distribuée, qui permet l'exploration des résultats et affiche des statistiques, telles que la persistance médiane et moyenne, le nombre d'occurrences de persistance par valeur, entre autres. Le client permet aux utilisateurs d'obtenir des résultats personnalisés et d'interroger des valeurs de persistance pour des spécificités d'intervalle, en reformulant la liste de la plus grande persistance, parmi diverses autres fonctionnalités.

Sommaire

1.Systèmes distribués et communication

Étude des systèmes distribués :

Nous utilisons un serveur avec le protocole telnet nous permettant de faire la liaison client-serveur-worker, générer et distribuer les tâches : Le serveur est chargé de générer les tâches à exécuter et de les distribuer aux workers. Il doit être capable de gérer les demandes de connexion des workers et des clients, de savoir combien de tâches chaque worker peut exécuter en parallèle et de distribuer les tâches en conséquence.

Suivre l'état des workers : Le serveur doit être capable de savoir quels sont les workers disponibles et de suivre leur charge actuelle. Cela permettra au serveur de répartir équitablement les futures tâches.

Stocker les résultats : Le serveur stocke les résultats obtenus par les workers. Il enregistre périodiquement les résultats sur le disque, ce qui permet de les sauvegarder en cas de perte de données.

Permettre la supervision du système : Le serveur est accessible par le client, qui peut consulter les résultats produits et visualiser des statistiques. Le serveur doit donc être capable de répondre aux demandes du client en fournissant les résultats souhaités.

Les outils de communication entre les machines :

Pour la communication entre les machines nous utilisons un service TCP/IP aussi appelé protocole de communication qui est utilisé pour transférer des données entre différents dispositifs connectés à un réseau, ici nous allons nous en servir pour communiquer entres des machines. Ce protocole va nous permettre :
-D'établir une connexion virtuelle.
-Se charge de maintenir l'intégralité de la communication.
-Gérer les erreurs de transmission.

Pour faire le lien serveur client nous utiliserons les classes Socket et ServerSocket

La première classe la classe Socket est utilisé pour créer une connexion réseau entre un client et un serveur distant. Lorsqu'un client veut se connecter à un serveur, il doit spécifier le nom d'hôte ou l'adresse IP du serveur ainsi que le port sur lequel le serveur écoute les connexions. La méthode Socket(serverName, serverPort) crée une nouvelle instance de Socket qui se connecte au serveur spécifié.

Voici un exemple :

```
String serverName = "localhost";
int serverPort = 8080;
Socket clientSocket = new Socket(serverName, serverPort);
```

La classe ServerSocket elle, est utilisée pour écouter les connexions entrantes sur un port spécifié ici 8080. Lorsqu'un serveur veut accepter des connexions de clients, il doit créer une instance de ServerSocket qui écoute sur un port spécifique. La méthode ServerSocket(serverPort) crée une nouvelle instance de ServerSocket qui écoute sur le port spécifié.

Exemple d'implémentation :

```
int serverPort = 8080;
ServerSocket serverSocket = new ServerSocket(serverPort);
```

2. Conception

Architecture du système :

L'architecture du système est composée de trois éléments principaux :

Le serveur : il est chargé de produire les tâches à exécuter et de les distribuer aux workers. Il gère également la communication avec les clients et stocke les résultats sur le disque.
Les workers : ce sont des machines distantes qui exécutent les tâches envoyées par le serveur. Ils se connectent au serveur pour lui signaler leur disponibilité et le nombre de tâches qu'ils peuvent exécuter en parallèle.
Les clients : ils se connectent au serveur pour consulter les résultats produits et visualiser des statistiques. Plusieurs clients peuvent se connecter au serveur en même temps.

Nous allons donc utiliser une architecture Maitre-Esclave :
- nos worker agiront comme tel : dès qu'une valeur possible à calculer arrive dans leur variable, ils lancent tous leurs Threads qui vont calculer jusqu'à épuisement de ce qu'on leur à envoyer.
- nos clients agiront comme tel : ils pourront envoyer des intervalles à calculer et visualiser le processus de calcul.

Détails d'implémentation :

Le serveur :
Il utilise des sockets pour communiquer avec les clients et les workers. Le serveur est capable d'accepter plusieurs demandes de connexions différentes. Les informations sur les workers disponibles sont stockées dans une table de hachage, qui est mise à jour en temps réel. Il est aussi à l'écoute des clients et traite leurs demandes. Le serveur utilise également une base de données pour stocker les résultats des calculs qu'il distribue aux workers connectés.

Les clients :
Ils se connectent au serveur via une connexion socket et peuvent envoyer des demandes pour consulter les résultats de calcul ou pour obtenir des statistiques sur les résultats (Moyenne, Médiane, Nombre d'occurrence par valeur persistance) ou encore la liste des nombres avec la plus grande persistance. Les clients peuvent également être utilisés pour demander un calcul sur l'intervalle de leur choix.

Les workers :
Ils se connectent au serveur via une connexion socket et envoient des informations sur leur disponibilité et leur nombre de cœurs. Les workers peuvent recevoir des tâches de calcul de la persistance multiplicative à partir du serveur, exécuter ces tâches en parallèle sur plusieurs cœurs et envoyer les résultats au serveur.

Les outils de stockage :
Pour le stockage des résultats de calcul, nous pouvons utiliser un simple fichier. Les résultats peuvent être stockés dans une table avec les informations suivantes : le nombre initial, la persistance multiplicative, la moyenne et la médiane. Les données peuvent être lues ultérieurement par le client pour générer des statistiques sur les résultats.

Choix des technologies utilisées :

Pour implémenter notre projet, chaque classe sera “extends” de Thread afin de pouvoir créer des files d’attente et un ordre de calcul.

Plus précisément, chaque worker va se connecter au serveur pour indiquer qu'il est disponible et spécifier le nombre de tâches qu'il peut exécuter en parallèle (nombre de cœurs disponibles). Ensuite, le serveur va distribuer les tâches de calcul à chaque worker disponible en fonction de leur charge de travail actuelle. Le worker va ensuite exécuter les tâches de calcul qui lui ont été assignées, c'est-à-dire calculer la persistance multiplicative de chaque nombre dans l'intervalle spécifié et renvoyer les résultats au serveur.

3. Réalisation

Description des étapes de développement :

Serveur :

Composition :

- 1 serverSocket : plateforme d’échange entre des Socket
- PrintWriter : permet d’écrire dans les consoles des Clients/Workers
- BufferedReader : permet de lire dans les consoles des Clients/Workers
- 2 HashTable :
    - Workers<Worker,Float> : stock les workers et leurs utilisations
    - Resultats<Integer,Integer> : stock périodiquement les résultats
- 1 tableau divisionTache : qui contient les intervalles à tester.
- 1 fichier Resultats.txt : pour conserver sur le disque les résultats
- 3 constantes de connexions :
- MAX_CLIENTS : nombre maximum de clients connectés
- MAX_WORKERS : nombre maximum de workers connectés
- MAX_TASKS_PER_WORKER : nombre de cœur maximum par worker

Démarrage du serveur :

Le serveur initialise ses attributs et écrit que sa mise en place est bonne et nous informe de son addresseIP ainsi que du port sur lequel la connexion doit se faire.
Il est alors en attente de connexion.

Client :

Composition :

- 1 Socket : c’est le socket qui est renvoyé lorsque la demande de connexion a été acceptée
- 1 id : pour savoir séparer les clients les uns des autres
- 1 pseudo : pour reconnaître ses commandes dans la console serveur
- PrintWriter / BufferedWriter : lire / écrire les commandes du client dans sa console

Démarrage / connexion au serveur :

- Le client initialise ses attributs et demande dans la console du client son pseudo. S’ensuit l’affichage des commandes disponibles :
- Calcul de la persistance des nombres sur un nombre ou un intervalle

- Une fois la demande écrite, elle est envoyée au serveur via le biais de la méthode Serveur.creerTache(début,fin)
- Affichage de la liste des nombres avec la plus grande occurrence
- Méthode : affichePlusGrandePersistance(id) avec l'id nécéssaire pour afficher ce résultat seulement dans la console du client qui le demande
- Statistiques : Moyenne/Médiane/Nombre d'occurrence par valeur de persistance
- Méthode : afficheMedianeMoyenne(id) et affichePersistance(id)
- Utilisation des workers
- Affiche la Hashtable où sont stocké les workers avec leur utilisation
- STOP : ferme le Socket donc la connexion

Worker :
Composition :

- 1 Socket : même utilisation que pour Client
- 1 id : pour le différencier des autres
- NbThreads : c'est le nombre maximum de cœur que le worker possède, en d'autres termes, c'est le nombre de Threads qu'il peut lancer en parallèle
- utilisation : c'est son pourcentage d'utilisation actuel
- ArrayList<Integer> intervalle : c'est la liste que le serveur remplira et que le worker devra toujours vider

Démarrage / connexion au serveur :
- Les attributs du worker sont initialisés comme chez les clients
- Le worker est start() par le serveur
- Il est ensuite en attente de travail et s'endort avec la méthode wait()
- Lorsqu'un client envoie une demande, il se peut qu'il reçoive des nombres à calculer dans sa variable interne intervalle qui doit être synchronisée vu que plusieurs Thread tenteront d'accéder à son contenu. Il est alors notifié et se remet en marche effectuant des calculs jusqu'à l'épuisement de tous les nombres dans la liste. Le worker est dans la capacité de lancer autant de Thread de calcul en parallèle que le serveur lui autorise. Une fois la liste vide alors tous les Threads créés devront se terminer (méthode join()) et le Thread principal se rendormira.
- A chaque fois que la persistance d'un nombre a été calculé, le worker renvoie ce résultat au serveur à l'aide de la méthode Serveur.ajouterResultat(nombre,persistance).

Traitement des résultats :
Une fois les résultats parvenus au serveur, il les enregistre dans la Hashtable Resultats en prenant soin de ne pas copier 2 fois le même résultat au cas où les Threads se seraient marchés dessus. Si cette table atteint la taille de 500 valeurs, on fait appel à la méthode ecrireResultat() afin de vider de l'espace mémoire. Cette méthode créer un FileWriter qui récupère l'addresse de notre fichier Résultat.txt. On parcourt ensuite toutes les valeurs de la Hashtable Resultats pour les écrire dans le fichier texte.
Une fois l'opération terminée alors la table est vidée. Cela ne signifie pas que les résultats sont perdus car ceux-ci sont aussi enregistrés dans un tableau persistance[] de taille 11 car c'est la persistance maximale jusqu'à 10^{333} , il s'incrémente alors à chaque résultat avec la persistance obtenue. Ce tableau est très utile pour calculer la moyenne et la médiane. Une ArrayList nommée NbGrandePersistance quant à elle contient tous les nombres avec la plus grande persistance obtenue dans l'échantillon calculé, elle est affichable ne permanence pour les clients.

Distribution des tâches :

Nous avons parlé du traitement des résultats et des calculs effectués mais pas de la distribution des tâches qui est primordiale.
A chaque fois qu'une tâche est ajoutée, le serveur va les distribuer. Pour cela, il découpe la demande (l'intervalle) avec le nombre de worker actuellement connectés dans le tableau divisionTache. Il va ensuite donner à chaque worker sa tâche à effectuer. Ce n'est clairement pas le modèle le plus efficace mais il est plutôt sécurisé et évite les problèmes de synchronisation ou de fuite mémoire.

4. Test et validation

Résultats :
1 CLIENT + (1 WORKER => 10 THREAD MAX) : 1.32s pour 50000 calculs (avec affichage)
2 CLIENTS + (1 WORKER => 10 THREAD MAX) : 36.75s pour 1000000 de calculs (avec affichage)
2 CLIENTS + (3 WORKER => 30 THREAD MAX) : 18.12s pour 1000000 de calculs (avec affichage)
3 CLIENTS + (6 WORKER => 60 THREAD MAX) : 1:55.42 pour 6000000 de calculs (avec affichage)

Les tests sont assez bons et concluant !

5. Conclusion

Bilan du projet :

En conclusion, ce projet nous a permis, d'aborder les notions des systèmes d'exploitation pour nous permettre de mettre en place un serveur distribué efficace pour le calcul de la persistance multiplicative des nombres. Grâce à la répartition des tâches sur plusieurs machines, nous avons pu bénéficier d'une grande capacité de calcul et traiter un grand nombre de nombres d'indices en un temps raisonnable. Même si certaines choses nous ont posé problème comme les liaisons entre les différentes structures, la synchronisation et l'ordonnancement notamment quand il s'agit d'accéder à une variable partagée.
Cependant, malgré les imperfections de notre code, nous sommes contents d'avoir réussis un système qui fonctionne dans sa globalité. A l'avenir, nous devrions repenser notre manière d'aborder les projets et éviter les erreurs de coordination et ne pas perdre le fil notre pensée lors de la construction du programme.

6. Annexes

Diagrammes :