SORBONNE UNIVERSITÉ

Réponse au question du projet Moteur d'exécution de workflows

Tarik Atlaoui

05 Mai 2020

1. Pourquoi ne peut-on pas utiliser le nom de la méthode en tant qu'identifiant de tâche ?

Si on utilisait le nom d'une méthode en tant qu'identifiant de tâche, l'utilisateur ne pourrait pas surcharger la méthode car deux méthodes avec le même nom mais des paramètres différent ne représenteraient qu'un seul nœud.

Exercice 3
2. Donner les grandes lignes de votre algorithme de la méthode execute. Il est attendu un algorithme décrit de manière synthétique et non un copier/coller de votre code.

Algorithm 1 Algorithm Exercice 3

```
1:  procedure EXECUTE(a, b)
2:      G ← Graph < String >
3:      C ← Context < String, Object >
4:      Mtab ← Method[]
5:      result ← Map < String, Object >
6:      for <node in Graph> do                          ▷ ont execute les method racine
7:          <Chercher la méthode qui correspond au node>
8:          if nodeParam is in C or nodeNbParam == 0 then
9:              result ← resultat
10:         else if si il y au moins une dépendance qui n'est pas dans C then
11:             Mtab ← methode                          ▷ la méthode de node
12:         else
13:             <invoke methode de node en recupérant les parametre dans C>
14:             <ou sans parametre>
15:         end if
16:     end for
17:
18:     for <m in Mtab> do          ▷ Ont execute les méthode qui présenter des dépendance
19:         <invoke m en recupérant les parametre dans G et C>
20:     end for
21:     return result
22: end procedure
```

Pour résumer l'algorithme utilisé, au debut en récupérer tous les résultats des tâches racine tout en gardant les méthodes des autres tâches qui ne peuvent pas encore être exécutées, puis parcourir les méthodes restantes et les exécuter.

Exercice 4
1. Décrire comment vous avez assuré le parallélisme des tâches indépendantes.

Nous avons considéré chaque nœud du graphe et lui avons dédié un thread. Il commence par chercher la méthode qu'il doit appeler, puis pour chaqu'un de ses arguments essaye de le recupérer, si ce dernier n'existe pas encore, il attend. Une fois qu'il a fini de recupérer la valeur de arguments, il invoque la fonction et met le resultat dans la map

Exercice 5

1. Décrire votre protocole de communication

Le JobExecutorRemote demande demande le calcule d'un job a une jvm distante puis il peux choisir celle ci lui renvois un tuple contenant le nombre de tache executer ainsi que la Map String , Object calculer

2. Justifier votre choix de l'API de communication que vous avez utilisée.

Nous avons choisi Rmi car elle paraissait être la plus adaptée au problème. Nous n'aurons pas besoin de gérer une base de donnée, il faudra juste gérer l'execution de tâche sur une ou plusieurs Jvm, elle est aussi incluse dans java donc pas besoin d'installation préalable ce qui est un plus.

3. Quelle hypothèse doit-on faire sur le type des objets du contexte ? Justifiez.

L'hypothèse serait que les types utilisés dans "Object" sont sérializable sinon on ne peut pas les transmettre d'une machine à une autre.

4. Quel mécanisme avez-vous utilisé pour la notification d'avancement au client ?

Dans le host qui fait le calcule il met une variable a false avants le calcule et a vrai apres des que la tache est fini il la place a true (l'idéal c'est de gardez une trace car si deux jobe son demander la notification peux etre pertuber)

Exercice 6

1. Quel est le protocole de communications entre maître-esclaves et éventuellement entre esclave-esclave ?

1-le Maitre reçoit la demande à qui il affecte un id et le renvoi au client
2- le client demande son résultat et se mets en attente si son résultat n'est pas encore prêt
3-le maitre lance un thread qui va chercher des esclaves à qui il demandera d'exécuter une tâche si possible
4-une fois trouvé, il lui passe le nom des tâches qu'il doit exécuter ainsi que le Job
5-une fois que l'esclave a terminé, il met la valeur de retour sur Taskmaster et décrémente un cpt dès que celui-ci est à 0 le client est libre de récupérer son Job

2. Comment le maître affecte ses tâches équitablement sur les esclaves ?

À chaque demande, il déplace l'index du premier esclave qui va recevoir la tâche

3. Comment les esclaves sont assurés de respecter leur borne de tâche courante ?

pour chaque tâche, l'esclave lance un Thread : Au début il decrémente la variable maxtask et la fin de son exécution il l'incrémente

4. Comment gérez-vous la communication d'un résultat entre deux tâches dépendantes ? Don- ner les avantages et les inconvénients de votre solution.

Les différentes tâches communiquent par le billet d'une map stockée sur le maitre. À chaque fois qu'une tâche se termine, elle dépose sa valeur de retour sur le maître, à l'inverse, si elle a besoin d'une valeur, il va l'y récupérer.

5. (niveau 2) Comment gérez-vous le fait d'avoir plusieurs jobs en cours d'exécution sur le cluster ?

le master ne fait que recevoir les jobs, le reste est fait par un thread pool

6. (niveau 3) Décrire le mécanisme qui permet de détecter la panne d'un esclave. Comment gérez-vous la réaffectation des tâches perdues ?
la panne d'un esclave est détectée lors de l'affectation des tâches, si l'esclave renvoi une connection exception c'est qu'il est en panne et par conséquent considéré par le thread qui affect les tâches comme tel pour le reste de l'exécution de ce thread. Les esclave qui avaient reçu un job reçoivent un signal d'abondon et on redemande toutes les tâches qui ne sont pas présentes dans map de resultat.