Rapport Final SAE3.02

Omar Chouffai
23/12/2024

Table des matières

1. Introduction

Ce rapport présente en détail le développement et l'utilisation d'un système de calcul distribué. Ce système repose sur un serveur maître qui gère les tâches envoyées par un client et les distribue à plusieurs serveurs secondaires appelés "workers".

Ce projet permet d'exécuter des programmes en Python, C et Java, tout en assurant une répartition automatique des charges et une gestion des pannes. Si aucun worker n'est disponible, le serveur maître peut exécuter les tâches localement.

---

2. Architecture globale

2.1 Vue d'ensemble

Le système est composé de :

1. **Serveur maître :**
    - Reçoit les tâches depuis le client.
    - Répartit les tâches aux workers disponibles.
    - Exécute localement les tâches si aucun worker n'est disponible.
2. **Workers :**
    - Reçoivent et exécutent les tâches envoyées par le maître.
    - Gèrent les tâches en Python, C et Java.
    - Se réactivent automatiquement après une panne.
3. **Client :**
    - Envoie les tâches au serveur maître.
    - Affiche les résultats reçus.

2.2 Diagramme de l'architecture

Client <-> Serveur Maître <-> Workers (5001, 5002, 5003)

Le client communique avec le serveur maître, qui délègue les tâches aux workers ou les exécute lui-même si nécessaire.

---

3. Technologies utilisées

3.1 Langages et outils

- **Python 3.x :** Pour le développement principal.
- **Sockets TCP/IP :** Pour la communication réseau.
- **GCC :** Pour compiler les programmes en C.

- **Java JDK :** Pour compiler et exécuter les programmes Java.
- **Git :** Pour la gestion des versions.

3.2 Bibliothèques utilisées

- **queue** : Gestion des files d'attente des tâches.
- **subprocess** : Exécution de commandes système pour compiler et exécuter des programmes.
- **socket** : Gestion des connexions réseau.
- **json** : Enregistrement des logs et des tâches inachevées.

---

4. Déploiement et tests

4.1 Déploiement

1. Installer Python, GCC et Java sur la machine.
2. Cloner le projet depuis GitHub.
3. Démarrer le serveur maître :

   python3 master_server.py

4. Démarrer les workers :

   python3 worker_server.py 5001

   python3 worker_server.py 5002

   python3 worker_server.py 5003

5. Lancer le client pour envoyer des tâches :

   python3 client_gui.py

4.2 Tests effectués

Test 1 : Avec un seul worker actif

- Vérifie si les tâches sont bien envoyées au worker.
- Simule une panne et vérifie le basculement vers le serveur maître.

Test 2 : Avec trois workers actifs

- Vérifie si les tâches sont réparties en alternance entre les workers (rotation après 3 tâches).

Test 3 : Exécution multi-langage

- Vérifie si les programmes en Python, C et Java sont bien exécutés.
- Nettoie automatiquement les fichiers temporaires.

Test 4 : Exécution locale sur le serveur maître

- Vérifie l'exécution locale quand aucun worker n'est actif.

---

5. Problèmes rencontrés et solutions

5.1 Réutilisation des workers

Problème : Les workers ne se réactivaient pas après avoir atteint leur limite de 3 tâches. **Solution :** Ajout d'un système cyclique pour les réactiver et reprendre les tâches après leur réinitialisation.

5.2 Gestion des pannes

Problème : Si un worker tombait en panne, les tâches étaient perdues. **Solution :** Mise en place d'une file d'attente et réaffectation automatique des tâches non terminées.

5.3 Multi-langage

Problème : Problèmes pour exécuter des programmes dans plusieurs langages avec des erreurs de compilation ou d'exécution. **Solution :** Gestion des commandes spécifiques à chaque langage et nettoyage des fichiers temporaires.

5.4 Exécution locale

Problème : Problème d'exécution locale sur certaines machines Linux. **Solution :** Utilisation explicite de commandes avec le préfixe `./` pour éviter les erreurs.

---

6. Améliorations possibles

1. **Interface Web :**
    - Ajouter un tableau de bord pour surveiller les workers et les tâches.
2. **Base de données :**
    - Enregistrer les tâches en cours pour éviter les pertes en cas d'arrêt du serveur.
3. **Sécurité :**
    - Chiffrer les communications entre les composants.
    - Ajouter un système d'authentification pour limiter les connexions.
4. **Optimisation des erreurs :**
    - Améliorer la gestion des logs et des messages d'erreur.

---

7. Conclusion

Ce projet montre comment construire un système distribué pour exécuter des tâches efficacement. Il gère les pannes et la répartition des charges de manière dynamique, tout en supportant plusieurs langages de programmation.

Les tests ont montré que le système est fiable et peut être utilisé comme base pour des développements futurs plus complexes. Des améliorations comme l'ajout d'une interface Web ou la gestion des tâches avec une base de données pourraient encore renforcer ses fonctionnalités.