RAPPORT FINAL

CLUSTER DE CALCUL – SAE3.02 - BUT RESEAUX ET TELECOMUNICATION

Gregory Runser

UNIVERISTE DE HAUTE ALSACE, IUT DE COLMAR

Préambule

Ce document a pour objectif de détailler l'architecture globale : ses fonctionnalités, le choix des technologies (Sécurité, Librairies), l'organisation du projet.
J'expliquerai également les défis rencontrés, les solutions mises en œuvre, et une réflexion sur les améliorations possibles pour le système.
Un second document détaillant la mise en place et le lancement des différentes composantes est disponible sur le répertoire GitHub.

Sommaire

Préambule 1
Sommaire 1
I. Partie Programmation 2
Fonctionnalités 2
Elémentaire 2
Avancé 2
Code Python 2
Client 2
Serveur Maitre 3
Choix des Technologies 4
Socket et Thread 4
Librairie externe 5
II. Partie Réponse 5
Aspect lié à la sécurité 5
Organisation 6
Diagramme de GANTT 6
Défis rencontrés et solutions mise en place 6
Choix de l'interface Graphique 6
Low-Balancing 6
Test sur les VM 7
Les améliorations possibles 7
Type de cluster 7
Une interface Graphique pour le serveur 7
Conclusion 8
Mon avis 8
Table des illustrations 8

I. Partie Programmation

Fonctionnalités

Elémentaire

Ce projet permet, depuis un Client/Hôte, l'envoie d'un code Python, Java ou C, à un serveur qui l'exécute et renvoie le résultat.

Avancé

Ce projet doit permettre un « Cluster de Calcul »[1], depuis donc plusieurs clients envoyer des codes a un serveur dit « Maitre » qui exécute le code et renvoie une réponse au client.
Cependant si le Serveur Maitre est « Surchargé », en termes de tache exécuter en simultané, il doit être capable de distribuer vers un Serveur dit « Esclave », qui prendra une charge de calcul, permettant de soulager l'intensité du serveur maitre.

Code Python

Client

Ce script est une application cliente avec interface graphique (GUI) développée à l'aide de PyQt5. Il permet de se connecter à un serveur, d'envoyer des fichiers, de recevoir des résultats, et de surveiller l'utilisation du CPU sur le serveur.

Fonctionnalités principales

- **Connexion au serveur** :
  - Entrer une adresse IP et un port pour établir une connexion avec le serveur.
- **Chargement et envoi de fichiers** :
  - Charger un fichier local.
  - Envoyer le fichier avec son contenu et son type (Python, C, Java) au serveur pour traitement.
  - Afficher le résultat renvoyé par le serveur.
- **Surveillance du CPU** :
  - Surveiller en temps réel l'utilisation du CPU sur le serveur.
  - Démarrer et arrêter la surveillance avec un bouton.

Structure du code – Classes Principales

- **CPUUsageThread** :
  - Thread pour surveiller l'utilisation du CPU sur le serveur.
  - Émet des signaux contenant les informations CPU reçues du serveur.
- **FileSenderThread** :
  - Thread pour gérer l'envoi de messages (incluant les fichiers) au serveur.
  - Émet des signaux contenant les résultats renvoyés par le serveur.
- **MainPage** :
  - Interface principale de l'application, regroupant les fonctionnalités d'envoi de fichiers et de surveillance CPU.
- **ConnectionPage** :
  - Interface de connexion où l'utilisateur peut saisir l'adresse IP et le port du serveur.

[1] Ensemble de plusieurs ordinateurs (ou nœuds), souvent interconnectés en réseau, qui travaillent ensemble pour exécuter des tâches de calcul intensif ou répartir des charges de travail.

- **ClientApp** :
    - Gestionnaire des différentes pages (connexion et interface principale).

Fonctionnement du Thread

- Les threads fonctionnent indépendamment pour ne pas bloquer l'interface graphique :
    - **CPUUsageThread** : Émet régulièrement des signaux contenant les données CPU reçues du serveur.
    - **FileSenderThread** : Émet un signal une fois que le message ou fichier a été envoyé et que le serveur a répondu.

Interface GUI (PyQt5)

- Les interfaces sont développées avec PyQt5 et comportent plusieurs widgets :
    - **Champs de saisie (QLineEdit)** : Pour l'adresse IP et le port.
    - **Zone de texte (QTextEdit)** : Pour afficher le contenu des fichiers et les résultats.
    - **Menu déroulant (QComboBox)** : Pour sélectionner le type de fichier (Python, C, Java).
    - **Boutons (QPushButton)** : Pour charger, envoyer des fichiers, et surveiller le CPU.

Serveur Maitre

Le **serveur maître** est capable de gérer et distribuer des tâches provenant des clients. Il peut exécuter des fichiers dans différents langages (Python, Java, C), surveiller l'utilisation du CPU, et rediriger des tâches vers des serveurs esclaves en cas de surcharge.

Fonctionnalités principales

- **Gestion des tâches en simultané** :
    - Le serveur gère un nombre défini de tâches en parallèle.
    - Si la limite est atteinte, les tâches peuvent être redirigées vers des serveurs esclaves.
- **Exécution de fichiers** :
    - Le serveur peut traiter des fichiers Python, Java, et C reçus des clients, les exécuter, et retourner les résultats.
- **Surveillance du CPU** :
    - Capable de calculer l'utilisation CPU du processus courant.
- **Redirection vers esclaves** :
    - Si le serveur maître est surchargé, il redirige les tâches vers des serveurs esclaves configurés.
- **Protocole réseau** :
    - Communication via sockets TCP.

Structure du code – Classes Principales

- **MasterServer**
    - Gère la réception des connexions clients.
    - Distribue les tâches entre les threads ou redirige vers les esclaves.
    - Attributs importants :

        - max_concurrent_tasks : Limite des tâches simultanées.
        - slave_ips : Liste des esclaves configurés pour la redirection.
        - current_task_count : Compteur de tâches en cours.
- **SlaveHandler**
    - Un thread pour chaque connexion client.
    - Gère l'exécution de la tâche reçue.
    - Fonctionnalités d'exécution :
        - Python : Exécute le fichier avec subprocess et retourne la sortie.
        - Java : Compile et exécute le fichier, puis retourne la sortie.
        - C : Compile avec GCC, exécute et retourne la sortie.

Choix des Technologies

Python permet d'exécuter une large gamme de codes, allant des plus simples aux plus complexes, en offrant une flexibilité pour différentes tâches. Dans ce projet, Python est utilisé non seulement pour gérer le lancement des interfaces clients, mais aussi pour exécuter des scripts sur les serveurs maîtres et esclaves.

Pour rendre cette solution encore plus robuste et fonctionnelle, plusieurs librairies sont nécessaires.

- Tout d'abord, Psutil est une bibliothèque essentielle pour surveiller la charge du processeur sur le serveur. Elle permet de collecter des informations détaillées sur l'utilisation du CPU, de la mémoire, ainsi que d'autres ressources systèmes, ce qui est crucial pour optimiser la performance du serveur.
- Ensuite, PyQt joue un rôle fondamental en permettant la création d'interfaces graphiques. Cette bibliothèque offre une approche structurée et modulaire pour concevoir des interfaces utilisateur, tout en permettant un design soigné et personnalisé.

Socket et Thread

- L'utilisation de Socket et Thread dans ce projet Python est liés à la gestion des connexions réseau et à l'exécution parallèle de tâches.

    - Un socket permet la communication entre différents programmes, souvent situés sur des machines distinctes dans un réseau. En Python, la bibliothèque socket fournit une interface pour établir des connexions réseau en utilisant les protocoles TCP/IP, UDP, etc.

      Communication entre clients et serveurs : Les sockets permettent d'établir une communication bidirectionnelle entre un serveur (qui écoute les connexions) et un client (qui initie la connexion). Cela peut être essentiel pour des applications distribuées, des services web, des systèmes client-serveur, etc.

    - Un thread permet d'exécuter plusieurs tâches en parallèle au sein du même programme. En Python, le module threading permet de créer et gérer des threads.

Les threads permettent d'effectuer plusieurs opérations simultanément sans bloquer l'exécution du programme principal. Par exemple, un programme peut gérer plusieurs connexions réseau en parallèle, où chaque thread répond à un client

- Pourquoi utiliser les deux ensembles ?
  L'utilisation combinée des sockets et des threads est très courante dans les systèmes de type serveur-client. Un serveur peut utiliser un socket pour écouter les connexions entrantes. Lorsqu'une connexion est établie, un nouveau thread est créé pour gérer cette connexion spécifique, permettant au serveur de continuer à écouter d'autres connexions sans interruption. Cela permet de gérer plusieurs connexions simultanément tout en maintenant la réactivité et l'efficacité du serveur.

Librairie externe

Ces libraires sont intégrés a python il m'a donc été possible de les utilisé. Seuls Psutil et PyQt sont à préinstaller.

- Psutil

Le module psutil (Process and System Utilities) fournit des outils pour surveiller et gérer les ressources système et les processus. Il est particulièrement utile pour obtenir des informations sur l'utilisation du CPU, de la mémoire, des disques, du réseau, etc.

- Uuid

Le module uuid permet de générer des identifiants uniques universels (UUID). Ces identifiants sont souvent utilisés pour garantir l'unicité dans des systèmes distribués, comme des bases de données ou des fichiers temporaires.

- Queue

Le module queue fournit des structures de file d'attente sûres pour les threads. Il est utilisé pour organiser et gérer des tâches dans ce projet ou il y a plusieurs threads.

II. Partie Réponse

Aspect lié à la sécurité

La sécurité est un aspect crucial dans la communication entre un serveur et un client.
En effet, bien qu'il soit possible d'implémenter une authentification dans un projet open source, cela peut devenir complexe, notamment parce que les informations peuvent être visibles en clair. La bibliothèque hashlib permet de sécuriser un mot de passe en le transformant à l'aide d'un algorithme de hachage, mais ce procédé, est facilement réversible.

Cependant, pour renforcer la sécurité, il est essentiel de ne pas seulement protéger les mots de passe, mais aussi de sécuriser les données échangées entre le client et le serveur. Cela peut être fait en utilisant des mécanismes de chiffrement, tels que des certificats SSL/TLS, qui garantissent la confidentialité des informations transmises. Le chiffrement via un certificat permet de sécuriser les échanges et d'assurer que les données ne soient pas interceptées ou altérées pendant leur transit.

Organisation

Diagramme de GANTT

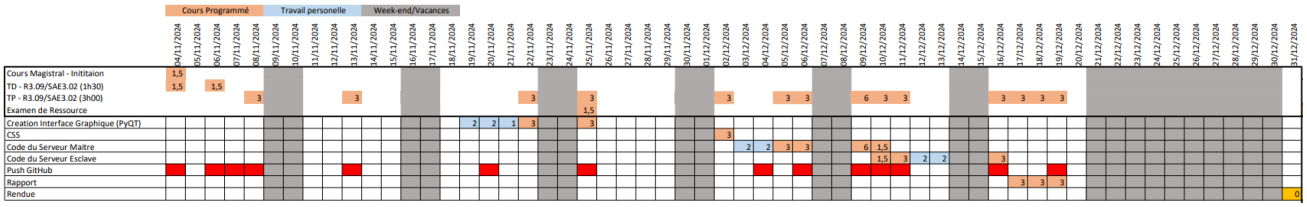

Figure 1 : Digramme de GANTT

TD pour ressource :	3
TD/TP pour Projet :	39
Heure alloué au projet :	39
Heure pour 'l'Int Graphique :	11
Heure pour le CSS :	3
Heure pour le Serveur Maitre :	17,5
Heure pour le serveur Esclave:	11,5
Heure pour l'ecriture des rapports :	9
	0
Heure TOTAL effectué pour le projet :	52

Heure réalisé en dehors des cours	13

Figure 2: Temps alloué au projet

Le nombre d'heure alloué a ce projet est conséquent, mais il m'a beaucoup plus, notamment sur la création de l'interface graphique, j'ai effectué des heures en dehors des heures dédié à la SAE pour avancer et avoir un rendue fonctionnelle, esthétique et développé.

Défis rencontrés et solutions mise en place

Choix de l'interface Graphique

Ayant, auparavant, déjà utilisé Tkinter, je me suis dit que ça serait à mon avantage de coder avec pour programmer l'interface client, cependant l'élégance de l'interface n'était pas à la hauteur de mes attentes ainsi au bout de 3 heures de conception avec Tkinter, j'ai basculé sur PyQt5. Plus élégant, tout aussi simple d'utilisation et complet ; il répondait à toutes mes attentes.

Low-Balancing

Au début, j'ai voulu faire en sorte que lorsque la charge de mon CPU sur le serveur maitre était supérieure à 70%, il redirige les taches vers un serveur esclave. Puis je me suis rendu compte de la complexité de faire augmenter l'utilisation du CPU, ainsi j'ai fait évoluer le low-balancing afin que la redirection se fasse non pas à l'utilisation du CPU, mais plutôt au nombre de taches en simultané.

Ainsi lors du démarrage il est nécessaire de renseigner en argument le nombre de tache courante maximum => (python3 master.py X) => X étant le nombre de tache en simultané que le serveur maitre peut exécuter avant de rediriger vers un serveur esclave

Test sur les VM

Etant étudiant, je n'ai que mon ordinateur portable ici à Colmar, pour effectuer des tests, ainsi il a été compliqué de lancer plusieurs VM en même temps dessus. J'ai dû effectuer les tests mais aussi la vidéo de présentation lors des pauses méridienne à l'IUT afin de disposer d'une infrastructure complète et performantes.

Les améliorations possibles

Type de cluster

Comme dit dans le document d'installation mon cluster a un AVANTAGE qui peut être un INCONVENIANT : il faut installer sur chaque serveur l'ensemble des dépendances nécessaire à l'exécution d'un programme, on ne peut pas faire de cluster sélectif comme ceci :

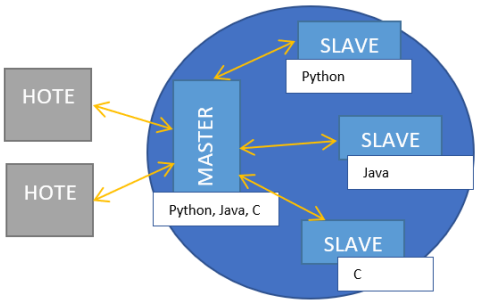

Figure 3 : Cluster Sélectif

Mais plutôt un cluster complet comme cela :

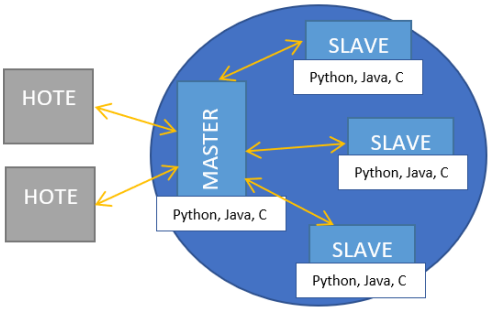

Figure 4 : Cluster Complet nécessaire pour ce projet

- L'avantage c'est que lorsqu'une VM Slave est formater (Dépendance installé), il suffit de la copier pour en avoir une seconde, changer son IP et la rajouter dans la liste du Serveur Maitre. De plus il n'est donc pas nécessaire de développé la redirection en fonction du type de fichiers.
- L'inconvénient est que chaque VM Slave est un peu plus lourde.

Une interface Graphique pour le serveur

Programmer une interface graphique pour le serveur maitre serait possible et permettrait de visualiser différentes informations tel que la charge du CPU mais aussi quelles taches est redistribuées vers les serveurs esclave.

Conclusion

Mon avis

J'avais beaucoup d'appréhension quant à ce projet, la programmation n'étant pas le domaine ou j'excelle le plus, je me demandais comment je ferais pour réussir, puis nous avons eu les cours de ressource qui m'ont vraiment motivé, j'ai trouvé de l'intérêt à faire ceci, et je me suis intéressé davantage à toutes les fonctionnalités (GUI, Thread et Socket) ainsi j'ai aimé faire cette SAE et le fait d'avoir de l'intérêt pour ce projet m'a poussé à donner le meilleur de moi-même.

Table des illustrations

FIGURE 1 : DIGRAMME DE GANTT 6
FIGURE 2: TEMPS ALLOUE AU PROJET 6
FIGURE 3 : CLUSTER SELECTIF 7
FIGURE 4 : CLUSTER COMPLET NECESSAIRE POUR CE PROJET 7