SAE 3.02 Rapport final

But final :

Avoir mis en place une architecture multi-serveurs avec serveur maître et cluster de serveur avec gestion de la répartition des charges dans le but de pouvoir gérer plusieurs clients en même temps de manière fluide et sans surcharge des serveurs.

Liste des taches :

- Etablir la connexion client-serveur
- Mettre en place la gestion client multiples
- Mettre en place la répartition de charge
- Mettre en place la communication (client-serveur, serveur-serveur)
- Fonctionnalités facultatives

Connexion client-serveur

But :

Etablir la connexion entre le client et le serveur via une interface graphique sur le client afin de pouvoir commencer la communication entre eux.
Cette connexion doit permettre l'envoi d'un programme python par le client, la réception, compilation et exécution du programme par le serveur

Mise en place :

La connexion entre les client et les serveur se fait via des sockets, le serveur maître étant toujours en écoute des requêtes de connexions sur le même port, la client a juste à indiquer ce port et l'adresse IP du serveur dans les zones associés de l'interfaces graphique puis il se connecte, à chaque fois qu'un client se connecte, le serveur accepte la connexion et démarre une thread sur laquelle il va se mettre en écoute de message, lorsqu'un message est envoyé par le client au serveur, le serveur le reçoit, détermine le language du code envoyé par celui-ci puis en fonction du langage, va envoyer le code brut au serveur esclave correspondant, il récupère ensuite le résultat envoyé par le slave et renvoie la réponse au client.

Gestion clients multiples

But :

Serveur doit être en capacité de gérer simultanément plusieurs client et requêtes en même temps.

Mise en place :

La possibilité du multi-client se fait à l'aide de threads qui vont permettre qui si un client se connecte, il ne bloque pas les autres connexions, on effectue donc une thread de la méthode d'acceptation du serveur maître ainsi que sa méthode de réception des messages, comme ça tout se fait de manière parallèle et sans blocage.

Répartition de charge

But :

Permettre la délégation de taches sur les serveurs esclaves par le serveur maître en cas de surcharge (limite du CPU ou nombre de programmes à gérer trop important), les serveurs esclaves doivent donc aussi être en capacité de pouvoir exécuter seul les programmes et retransmettre le résultat au serveur maître.

Mise en place :

Pour ce qui est de la répartition de charge, malheureusement j'ai eu des problèmes avec les multi-serveurs, il y a quand même une implémentation d'une limite de programme et de limite de CPU, mais celle-ci ne peut que bloquer les nouvelles compilations à faire et non pas à les déléguer comme prévu, il ne pourrait que le déléguer au serveur maître mais pas à un autre serveur slave. Pour ce qui est de la compilation, chaque serveur compile et interprète lui-même les programmes en fonction de leur langage attitré.

Communication des équipement

But :

Via la création de sockets, établir la communication entre le client et le serveur et les différentes connections entre les serveurs du cluster.
Cette partie permettra aussi la mise en place des différentes gestion d'erreurs, et différent éléments nécessaire à la fiabilité et robustesse des connexions.

Mise en place :

La connexion entre les serveurs esclaves et le serveur maître se fait de la même façon que la connexion client-master, le master écoute, les slaves se connecte, mais à l'aide d'un script implémenter dans le master, les adresses IP des serveurs esclaves sont attitrés à un langage de programmation, en effet cette connexion permet notamment de pouvoir transférer les éventuelles erreurs du serveur slave au client, afin que celui-ci ne soit pas bloqué infiniment dans une boucle en attentant une réponse par exemple ou alors simplement transmettre les erreurs de compilation et même tout bonnement leur résultat.

Fonctionnalité supplémentaires :

But :

Mettre en place les différentes fonctionnalités pouvant être ajouté afin d'augmenter la fiabilité, sécurité et la qualité globale du service. Pour ce faire voici une liste de fonctionnalités pouvant être mis en place :
- Le monitoring du cluster
- La persistance des données
- la sécurité
- la scalabilité
- la robustesse

Problèmes et solutions :

Les différents problèmes que j'ai pu rencontrer au cours de ce projet sont principalement les erreurs liées aux thread, notamment au niveau de l'affichage et des mises à jour des différents blocs du GUI qui ne supportent pas d'être modifié hors de la thread principale, dans certains cas je n'ai pas su trouver de solutions (comme par exemple dans mon fichier slave.py, l'affichage d'une représentant le pourcentage de CPU à chaque réception d'un nouveau code à compiler afin d'effectuer un monitoring du serveur, malheureusement soit la progression se mettait une fois puis plus jamais de mise à jour, soit elle fonctionnait parfaitement mais au bout d'une dizaine de secondes, l'interfaces graphique crashait complètement malgré que le serveur soit toujours opérationnel et puisse compiler en arrière plan, seul l'interface graphique avait crash.).

Un des autres gros problèmes est la déconnexion des serveurs esclaves qui ne se fait

pas, malgré plein de tentative différentes, je n'ai jamais réussi à faire en sorte de pouvoir déconnecter les serveurs esclaves sans qu'ils crash.
L'arrêt des programmes est aussi un défis que j'ai pu rencontrer et que je n'ai pas su régler.

La découverte des différentes compilation des programmes était aussi une petite difficulté qui a demander un peu de recherche, notamment pour Java.

La principale amélioration que je pourrais et devrait faire sur mon projet serait d'augmenter sa stabilité, afin de faire en sorte qu'elle évite le plus les erreurs possibles, ainsi que l'affichage de plus d'informations, du coté graphique il pourrait y avoir un peu plus d'informations afin de comprendre réellement comment ça fonctionne et ce qui se passe en cas de problème (pour ne pas à avoir à passer par l'interfaces ligne de commande)

Sources

Voici ici la liste de toutes les sources que j'ai utilisé en plus du cours durant ce projet et qui m'ont pour la plupart bien aidé :

Regex :
- https://www.w3schools.com/python/python_regex.asp
- https://regex101.com/

PyQT :
- https://www.pythontutorial.net/pyqt/
- https://www.pythontutorial.net/pyqt/pyqt-qfiledialog/
- https://www.pythontutorial.net/pyqt/pyqt-qpushbutton/
- https://www.tutorialspoint.com/pyqt/pyqt_qcombobox_widget.htm
- https://stackoverflow.com/questions/23835847/how-to-remove-item-from-qlistwidget
- https://www.geeksforgeeks.org/pyqt5-toggle-button/

CSS dans PyQT :
- https://stackoverflow.com/questions/64835508/multiple-colors-in-background-transition

Subprocess
- https://www.digitalocean.com/community/tutorials/how-to-use-subprocess-to-run-external-programs-in-python-3-fr
- https://stackoverflow.com/questions/76090257/run-c-file-with-input-from-file-in-python-subprocess-library

Compilation C / C++ :
- https://www.tutorialspoint.com/cplusplus/cpp_environment_setup.htm
- https://www.tutorialspoint.com/how-to-compile-and-run-the-cplusplus-program

Compilation Java :
- https://askubuntu.com/questions/145748/how-to-compile-a-java-file-on-ubuntu