C-WIRE

PREING 2 – 2024/2025

Réalisé par :

AGBAVOR Emmanuel
BRINGUIER Valérian
EDWARS Noam

Enseignant :

ANSERMIN Eva
GRIGNON Romuald

Répartition :

Afin d'éviter tout conflits dans le GitHub entre plusieurs versions d'avancement du projet nous avons décidé de toujours avancer et coder tous les trois ensembles, quelquefois réunis en présentiel à CY-TECH mais la plupart du temps chacun chez nous en appel vidéo. Ainsi seule une seule personne poussait les avancements du code sur le GitHub.

C'est avec cette approche de réflexion commune face à chaque nouvelle étape et problème du sujet que nous avons réussi à avancer pas à pas jusqu'à la réalisation complète du projet et de son cahier des charges.

Planning de réalisation :

Afin d'organiser notre travail, nous avons d'abord revu l'utilisation du langage Git parallèlement avec l'application GitHub afin d'être opérationnels sur le dépôt du code, changer de branche, régler des problèmes de commit…

Nous avons donc procédé par plusieurs étapes que nous allons détailler par la suite :

1/ Point d'entrée de l'utilisateur

Les premières étapes du programme commencent par le shell : vérifier les paramètres d'entrée, créer la fonction d'aide et lancer la compilation du programme C.

2/ Filtre des données

L'utilisateur a la possibilité de renseigner les informations qu'il souhaite analyser et être retranscrites dans les fichiers de sortie. Pour ce faire nous procédons donc à un filtre et un formatage des données sources au sein de notre fichier shell. Ce filtre permet de réduire drastiquement la quantité de données qui sera traitée par la suite.

2/ Création de l'arbre

Une fois en possession de nos données nouvellement filtrées et formatées, Nous pouvons donc créer un arbre AVL. Au fur et à mesure que nous construisons l'arbre, nous mettons à jour les consommations réelles des différents nœuds afin d'avoir un temps d'exécution réduit. Cela se retranscrit par deux fonctions dans notre code C, l'une (*parserLigneCSV*) qui permet de venir lire nos données sources, et une seconde (*creerArbre*) qui a pour objectif de créer l'arbre AVL.

Initialement, nous avions prévu de laisser l'entièreté du traitement des données au code C, mais après différents échanges et réflexions, nous avons opté pour le comportement décrit précédemment.

3/ Ecriture des données

Une fois notre arbre construit et mis à jour, nous pouvons alors persister ces informations dans des fichiers.

Il existe alors deux fonctions d'écriture :

- Écriture des données génériques d'une station, à savoir sa capacité et sa consommation réelle
- Écriture des données propres aux postes LV, avec en plus de leur capacité et consommation, la différence absolue entre ses valeurs.

4/ Tri des données

Le tri des données finales se fait au travers du script shell : celui-ci va alors lire les fichiers générés par le code C.

Un traitement supplémentaire est également réalisé dans le cas où les stations analysées sont les postes LV. Un nouveau fichier sera alors créé afin de récupérer les informations des dix plus grandes stations en surconsommation et des dix plus petites stations en sous-consommation de ces stations.

5/ Réalisation des graphiques

La dernière partie de notre exécutable shell consiste en la réalisation de graphiques. Pour ce faire nous avons utilisé l'utilitaire *Gnuplot.* Cette étape a commencé par une plongée approfondie dans la documentation pour appréhender les fondements de ce « nouveau » langage. Une fois ces bases assimilées, nous avons créé un graphique structuré en plusieurs colonnes, chacune représentant une station, accompagnée de légendes dynamiques qui s'adaptent constamment aux requêtes de l'utilisateur."

Ici aussi, deux types de graphiques existent :

Le premier, générique, permet d'afficher simplement la consommation et capacité des stations analysées sous forme de colonnes.

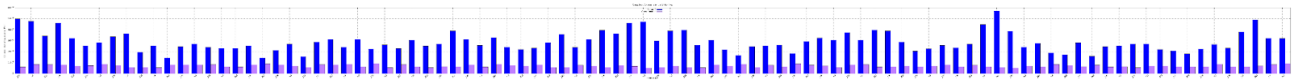

FIGURE 1 : CAS DU TRAITEMENT DES STATIONS HVB COMP

Le second est destiné à l'affichage des postes LV min et LV max avec notamment leur information absolue de sur ou sous consommation. Celui-ci reprend les informations du graphique générique en plus d'une troisième colonne, afficher en vert lorsque la station est en sous consommation ou bien afficher en rouge lorsque celle-ci est en surconsommation.

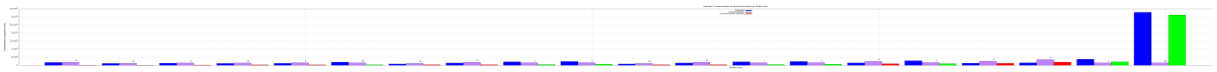

FIGURE 2 : CAS DU TRAITEMENT DES STATIONS LV ALL

5/ Temps de traitement

Différents temps de traitement effectifs sont également enregistrés et partagés à l'utilisateur. On retrouve :

- Le temps de filtrage des données sources
- Le temps d'exécution du programme C comprenant la création de l'arbre AVL ainsi que l'écriture des résultats
- Le temps de tri de ces résultats
- Le temps de génération des graphiques

Limitations fonctionnelles :

Certaines limitations fonctionnelles sont présentes :

- Le temps de traitement de la création des graphiques n'est pas optimal car le programme va recréer un graphique pour chaque résultat de demande du client déjà présent dans le dossiers /tests. Pour pallier cela nous pourrions alors vider le dossier /tests entre chaque utilisation du programme par le client ; cependant nous perdrions alors les données déjà calculées. Ainsi il sera préférable de corriger le script shell afin de seulement créer le graphique de la demande actuelle.
- En raison des différences du nombre de stations de différents fichiers, l'échelle de nos graphiques varient et donc l'interprétation de ces derniers peut s'avérer difficile. Il faudrait alors adapter la taille et l'échelle des graphiques automatiquement en fonction des entrants reçus. Changer le type d'échelle (linéaire, logarithmique) peut être un axe d'amélioration.
- Au vu du nombre de données présentes lors de l'analyse de poste LV, nous avons fait le choix de pas générer les graphiques découlant des analyses complètes ; l'affichage y étant illisible pour un être humain