Rapport de projet : C-WildWater

Informatique

Réalisé par :

CORIOU Etienne - JEUDY Martin - SBAI Sayf

2025/2026

Enseignant : Romuald Grignon

Sommaire

Introduction

Le projet C-WildWater nous a plongés dans un défi de taille : le traitement de données de masse appliqué à la gestion de l'eau. Notre objectif était de bâtir un système capable de modéliser et d'analyser les flux d'un réseau complexe, reliant les sources de captage aux usagers finaux, en passant par les usines et les réservoirs. Réussir à transformer 8 millions de lignes de données brutes en un programme efficace a nécessité une coordination et une communication dans notre trinôme.

Ce rapport retrace notre parcours à travers quatre parties. Nous commencerons par vous détailler l'organisation que nous avons mise en place pour structurer notre équipe. Nous expliquerons ensuite nos choix techniques nécessaires à la réalisation du projet. Puis, nous reviendrons sur notre planning de réalisation, avant de conclure par une présentation de nos résultats visuels, sans oublier les limites rencontrées et les leçons que nous avons tirées de ce projet.

I - Organisation du groupe

La réussite de ce projet reposait sur une collaboration étroite entre les membres du groupe, compte tenu de la dualité technique imposée par le sujet : le traitement de flux via Shell et le calcul algorithmique en C. Nous avons choisi de diviser notre projet en plusieurs tâches à répartir entre les 3 membres afin de faciliter la réalisation du projet.

Sayf s'est occupé de mettre en place la base du programme. Il a créé les structures de données principales qui permettent de stocker proprement les informations du réseau. Il a surtout travaillé sur la partie AVL, qui était obligatoire pour ce projet.

Martin a travaillé sur la lecture du fichier et les algorithmes de calcul. Il a fait le Parser, qui sert à lire le fichier CSV ligne par ligne et à découper les informations pour les donner au reste du programme. Il a également codé la partie sur le calcul des fuites. C'est un algorithme qui doit parcourir tout le réseau à partir d'une usine pour faire le cumul de tout ce qui est perdu dans les tuyaux jusqu'aux usagers.

Etienne a mis en place toute la logique de vérification des arguments, s'assurant que l'utilisateur saisit des commandes valides et que les fichiers nécessaires sont présents. Il a également automatisé la gestion de la compilation : son script est capable de détecter l'absence de l'exécutable et de lancer le Makefile de manière autonome. Un autre aspect important de son travail a été l'intégration de Matplotlib en Python. Il a rédigé les scripts permettant de transformer les données de sortie brutes en histogrammes PNG clairs et exploitables. Il a également fait le module histogramme.c pour récupérer les données (max, src, real, all) et les écrire dans les fichiers de sortie.

II – Choix d’architecture et de conception

Pour que notre programme soit capable de traiter 8 millions de lignes rapidement, nous avons dû séparer le projet en trois outils qui communiquent entre eux :

1. Le script Shell : Il prépare le terrain (validation des arguments, affichage du message d’aide, vérification des fichiers, vérification de l’environnement, mesure du temps d’exécution) et transmet les arguments au programme C
2. Le programme en C : Étant rapide, c'est lui qui contient toute la logique mathématique, le parsing, et l’insertion des valeurs dans des AVL.
3. Le script Python : Il permet d’exporter les données traitées sous format .png grâce à la bibliothèque matplotlib.

La pièce maîtresse de notre code est l'arbre AVL. Nous l'utilisons pour stocker tous les éléments du réseau (usines, stations, clients).

- La fonction find_or_create_component permet de chercher ou d'ajouter un élément très vite.
- Le tri : Au lieu de trier la liste à la fin, nous avons utilisé une astuce lors de l'insertion. Dans la fonction insert_facility, nous avons codé la logique de rangement pour que les noms soient déjà ordonnés.
- Le parcours inverse : Pour l'affichage, nous utilisons la fonction print_avl_reverse. En parcourant l'arbre via un parcours infixe inversé, on récupère les données directement dans le bon ordre.

Pour calculer les pertes d'eau, nous avons modélisé le réseau comme un arbre généalogique :

- La fonction chargerReseauLeaks lit le fichier et crée des liens entre les composants "parents" et leurs "enfants" grâce aux pointeurs first_child et next_sibling.
- Le calcul lui-même est géré par la fonction calculate_recursive_volume. C'est un algorithme qui part d'une usine et "descend" récursivement dans tous les tuyaux. À chaque étape, il divise le volume d'eau et soustrait le pourcentage de fuite.

Enfin, nous avons utilisé le mode "Append" pour l'écriture des résultats. Cela permet de rajouter les nouveaux calculs à la suite dans le fichier leaks.dat sans effacer les tests précédents, ce qui est très utile pour garder un historique.

III - Planning de réalisation

Pour mener à bien ce projet dans les délais impartis, nous avons organisé notre travail suivant un calendrier précis.

Semaine	Objectifs principaux	Tâches réalisées	Livrables
Semaine 1	Prise en main du sujet et conception	- Lecture et analyse du sujet - Compréhension du format CSV et de la topologie du réseau - Répartition des tâches au sein du groupe - Mise en place du dépôt GitHub - Début du script Shell (gestion des arguments) - Début du Makefile	- Dépôt GitHub initialisé - Structure du projet prête
Semaine 2	Implémentation des fonctionnalités principales	- Parsing du fichier CSV en C - Implémentation des AVL pour les usines - Calcul des volumes (max, capté, réel) - Génération des fichiers de données pour les histogrammes- Appel du programme C	- Programme C pour les histogrammes en cours de réalisation - Commits réguliers sur GitHub

		depuis le script Shell - Gestion des erreurs principales	
Semaine 3	Finalisation, tests et rendu	- Construction de l'arbre de distribution aval - Calcul des fuites pour une usine donnée - Génération du fichier historique des fuites - Génération des graphiques PNG - Tests complets - Nettoyage du code et commentaires - Rédaction du rapport PDF et du README	- Application complète et testée - Dossier tests rempli - Rapport PDF final - Dépôt GitHub prêt pour le rendu

IV - Présentation des résultats et limites rencontrées

Le script Python récupère les données classées par identifiant décroissant, et les trie par volumes croissants.
Certains graphiques contiennent des valeurs assez proches. Pour mieux les différencier, nous avons décidé d'utiliser un type de données double dans la structure Facility pour une meilleure précision.

Figure 1 : Histogramme des 50 plus petites usines (capacité maximale)

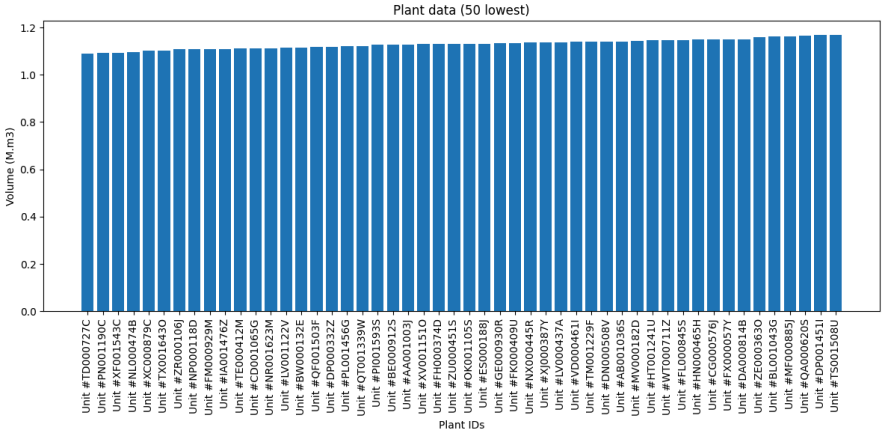
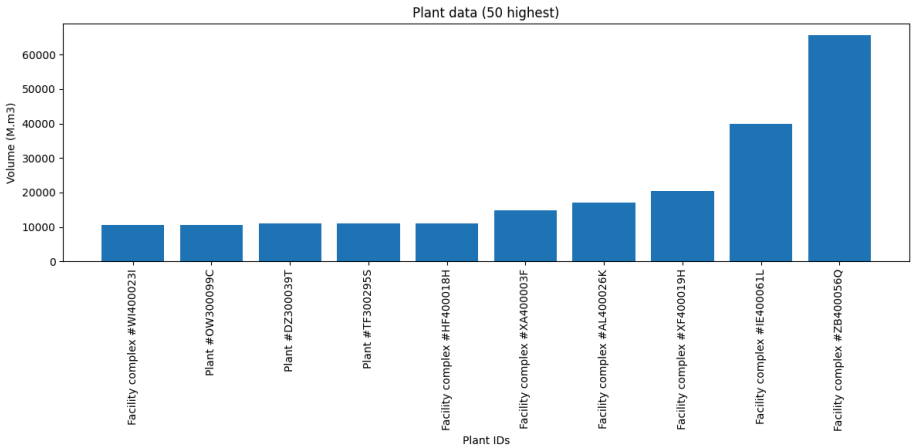


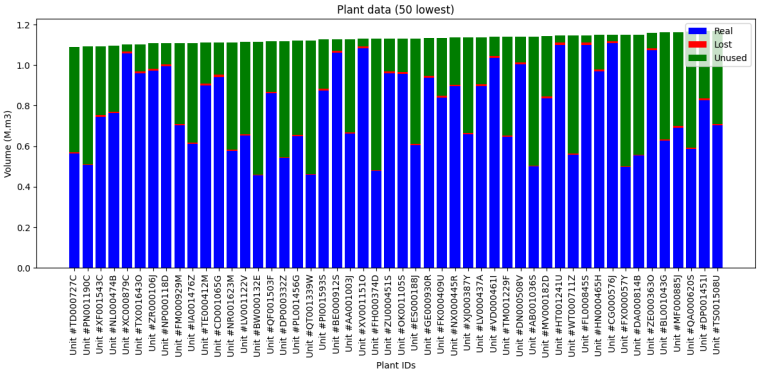
Figure 2 : Histogramme des 10 plus grandes usines (capacité maximale)



Ici, c'est la même chose mais pour les 10 plus grandes usines.

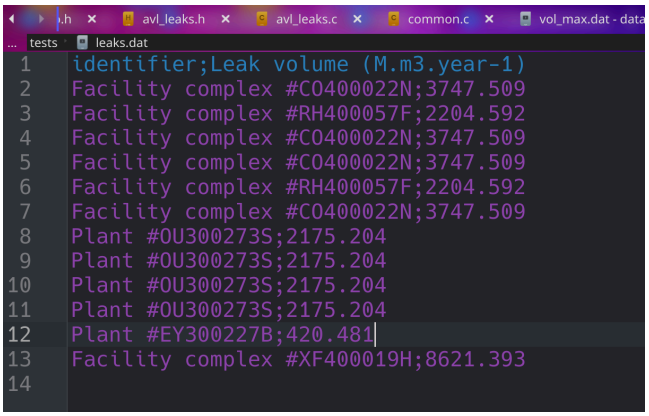
Enfin, ce dernier type d'histogramme 'all' permet d'afficher les 3 types 'max', 'src', et 'real' dans une même image, pour avoir une vision globale des données.

Figure 3 : Histogramme des 50 plus grandes usines (volume réel, perdu, inutilisé)



Enfin, ce fichier est le résultat direct de la fonction leaks. On y voit l'identifiant de l'usine suivi du volume perdu calculé par calculate_recursive_volume. L'affichage en millions de mètres cubes permet une lecture rapide des pertes totales.

Figure 3 : Fichier leaks.dat rempli avec quelques valeurs



Nous avons malgré tout rencontré quelques limites :

Premièrement, nous avons eu beaucoup de mal à comprendre l'énoncé du projet au premier jour. Les structures étaient assez complexes et le cahier des charges était plutôt long.

De plus, nous nous sommes rendu compte que le projet était axé sur deux grandes fonctions: l'histogramme et les fuites. Or, vu que nous étions trois, il était assez compliqué d'avancer au début, par peur d'empiéter sur le code d'un autre membre du groupe. Nous avons donc laissé Étienne s'occuper du script Shell et du Python, avant d'avoir un programme qui pouvait générer des données.

En outre, le programme devait lire un fichier de plus de 8 millions de lignes et les insérer dans un AVL, avec autant d'appels récursifs. Le moindre oubli de libération de mémoire ou de condition d'arrêt entraînait une fuite gigantesque, pouvant faire planter nos ordinateurs. C'est arrivé une fois, et la consommation de RAM est montée à environ 20GiB !

Aussi, il a été très difficile d'obtenir un temps d'exécution le plus faible possible. Malgré avoir essayé d'optimiser nos fonctions, nous avons discuté avec d'autres groupes et nous sommes rendu compte que notre programme était plus lent que d'autres en exécutant la commande leak.

Conclusion

Malgré les difficultés rencontrées, notamment pour stabiliser la gestion de la mémoire sur un volume aussi massif de données et pour dompter la structure complexe de l'arbre AVL, nous avons réussi à mettre au point un programme robuste et plutôt rapide.

Cette réussite ne se mesure pas seulement à la justesse des calculs de fuites ou à la réussite des histogrammes produits, mais également à notre capacité à collaborer efficacement. Nous avons su surmonter les difficultés rencontrées au cours de ce projet.

En conclusion, ce projet C-WildWater nous a prouvé qu'avec une organisation solide et une bonne dose de persévérance, il est possible de traiter des problématiques de "Big Data" avec les outils vus en cours, tout en restant précis et performant.