Rapport Final - Projet Hagimule

Binôme : Hermas OBOU, Corentin COUSTY

Réalisations Techniques

1. Module Diary

- **Gestion dynamique des clients** :
  Le Diary enregistre et maintient à jour les informations de chaque client connecté, incluant son nom, l'adresse de son daemon, et sa fréquence d'utilisation. Cette gestion garantit une vue globale et actualisée du réseau de clients disponibles.

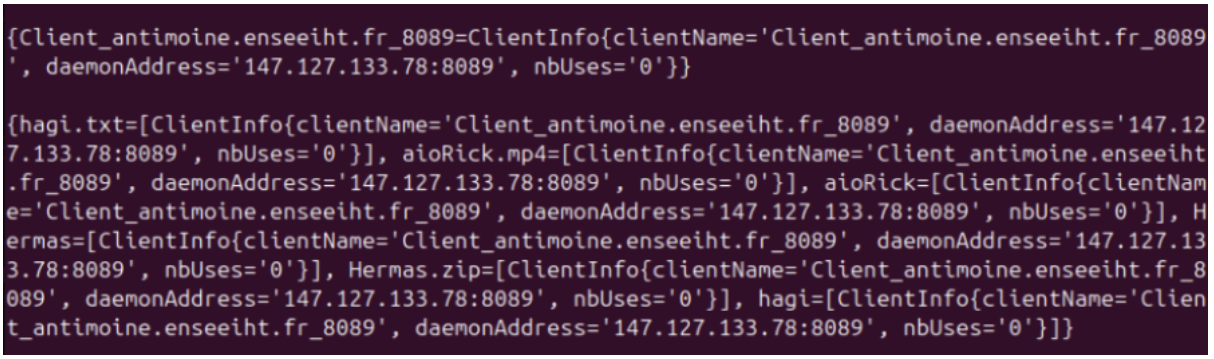
2. Assemblage du Downloader et du Daemon

- **Téléchargement simultané** :
  Les fichiers sont découpés en fragments indépendants. Ces fragments sont téléchargés simultanément depuis plusieurs sources, ce qui améliore considérablement le débit global.

3. Base de Données Centralisée

La base de données est un composant clé de la gestion des fichiers et des informations associées :

- **Stockage des checksums** :
  Les checksums sont utilisés pour identifier de manière unique chaque fichier. Cela élimine le besoin de recalculer systématiquement les checksums, notamment pour les fichiers volumineux, ce qui réduit la charge de traitement.
- **Enregistrement des types de compression** :
  Chaque fichier est associé à son type de compression spécifique, une information transmise au downloader pour garantir que les fragments téléchargés sont décompressés correctement et efficacement.
- **Identification unique des fichiers par checksum** :
  La base de données utilise les checksums pour identifier et localiser les fichiers détenus par les différents daemons. Cela assure une traçabilité et une cohérence dans les transferts de fragments entre clients.



4. Vol du Travail via un Buffer Ring

Mise en Place du Vol du Travail

Pour gérer efficacement les fragments à télécharger et assurer la continuité du processus, nous avons implémenté un système reposant sur **deux queues** distinctes :

1. **Queue des fragments à traiter** :
    - Contient les fragments de fichier encore non assignés.
    - Les daemons se servent dans cette file pour récupérer un fragment disponible.
2. **Queue des fragments en cours de traitement** :
    - Une fois qu'un daemon récupère un fragment dans la première queue, ce fragment est déplacé dans la **queue des fragments en cours de traitement**.
    - Cette file stocke les fragments déjà assignés, mais non encore finalisés.

Fonctionnement du Vol du Travail

1. **Attribution Initiale** :
    - Les daemons commencent par consulter la **queue des fragments à traiter** pour y récupérer un fragment disponible.
    - Dès qu'un fragment est assigné, il est déplacé dans la **queue des fragments en cours de traitement**.
2. **Passage Automatique à la Deuxième Queue** :
    - Une fois la **queue des fragments à traiter** vide, les daemons se tournent vers la **queue des fragments en cours de traitement**.
    - Ils récupèrent les fragments encore présents dans cette file, ce qui indique que :
        - **Le fragment n'a pas été finalisé par le daemon initial.**
        - **Il reste à traiter et peut être repris par un autre daemon.**
3. **Vol de Fragments** :
    - Lorsque des fragments sont disponibles dans la queue des fragments en cours de traitement, les daemons actifs se les attribuent automatiquement, en fonction de leur disponibilité.
    - Cette réattribution garantit que les fragments sont toujours traités, même si un daemon initial est lent ou en panne.

4. **Équilibrage Dynamique** :
    - Les daemons qui terminent rapidement leur traitement reviennent chercher des fragments dans la queue en cours de traitement, prenant ainsi en charge les fragments restants ou bloqués.
    - Cette approche équilibre la charge entre daemons, réduisant les temps d'attente.

Résilience aux Pannes

Ce système en deux queues est intrinsèquement résilient :

- **Aucune perte de fragments** : Les fragments restent dans la queue des fragments en cours de traitement tant qu'ils ne sont pas finalisés, garantissant qu'ils seront toujours traités par un daemon.
- **Adaptation aux pannes** : En cas de panne ou de déconnexion d'un daemon, les fragments qu'il n'a pas finalisés sont automatiquement repris par d'autres daemons.
- **Vol de travail efficace** : Les daemons disponibles réagissent immédiatement aux fragments restants, assurant ainsi une continuité du traitement sans intervention manuelle.

🙂 : Petit point amusant à la toute fin tous les daemons foncent vers le dernier fragment qui n'est pas encore traité pour le traiter 🙂

5. Gestion de la Latence et Lancements Multi-Plateformes

- **Simulation de latence réseau** :
  Une gestion configurable de la latence permet de simuler des conditions variées.
- **Compatibilité multiplateforme** :
  Le système est fonctionnel en local et sur les machines de l'école

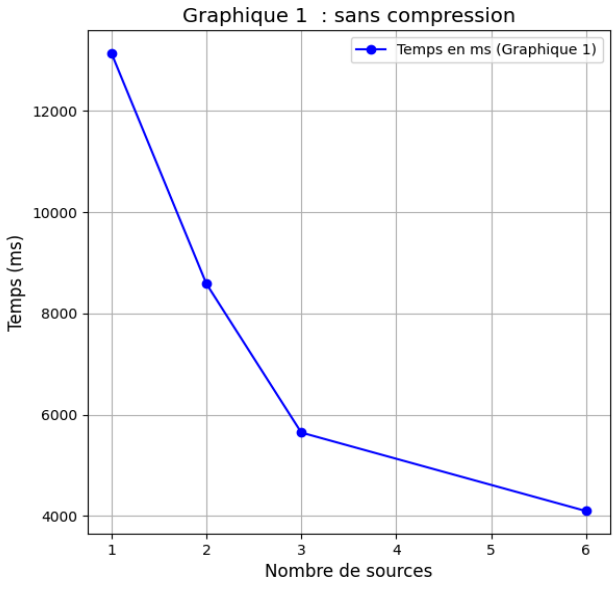
Répartition des Tâches

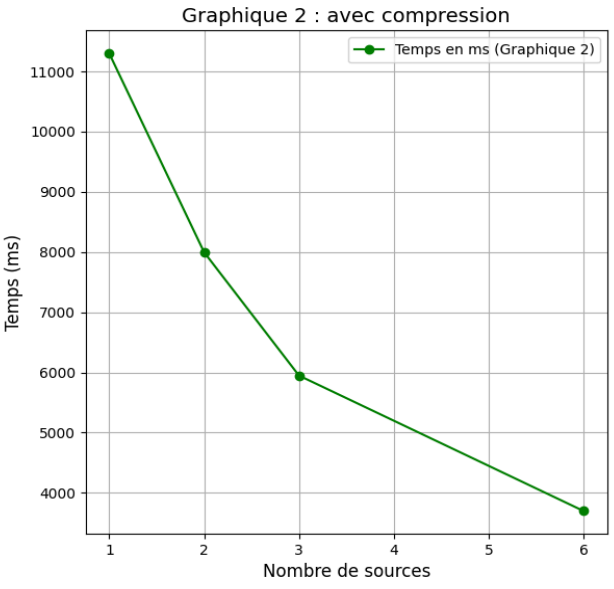
Sujet	Propriétaire	État
Architecture générale	hermas OBOU	Approuvée
Module Diary	Hermas OBOU	Approuvée
Intégration de la base de donnée	Corentin COUSTY	Approuvée
Vol de Travail	Corentin Cousty	Approuvée
Résilience et Compression	Hermas OBOU Corentin Cousty	Approuvée
Scripts et gestion de la latence	Corentin Cousty	Approuvée
Test et Évaluation	Hermas OBOU	Approuvée

Évaluation des Performances

Résultats des Tests

Les résultats sont présentés pour un fichier de 64 Mo téléchargé depuis plusieurs sources, avec deux scénarios : **sans compression** et **avec compression (algorithme Zstandard)**. Les temps mesurés incluent le traitement global, intégrant la compression, la décompression et la transmission réseau.





Analyse des Résultats

1. **Scénario sans compression :**
    - Le temps de téléchargement diminue à mesure que le nombre de sources augmente, démontrant les avantages du parallélisme.
    - Pour 6 sources, le temps est divisé par un facteur supérieur à 3 par rapport à une source unique.
2. **Scénario avec compression :**
    - La compression réduit le volume des données transmises, mais les gains sont affectés par le temps nécessaire à la compression et à la décompression.
    - Pour un fichier de 64 Mo, le surcoût du traitement réduit les bénéfices apportés par la compression. Et ce alors qu'on compresse les fichiers en avance.

Perspectives d'évolution

1. **Adapter l'algorithme de compression**
    - Différents algorithmes de compression ont été implantés, on peut les choisir via la ligne de commande pour le moment mais on pourrait aussi adapter l'algorithme de compression au fichier.
    - On pourrait ainsi gagner du temps car la décompression peut prendre du temps.
2. **Mesurer le débit en temps réel dans le registre**
    - Pour pouvoir mesurer l'utilisation et la capacité de chaque machine très précisément et ainsi récupérer les adresses disponibles dans l'ordre croissant de leur disponibilité.

Conclusion

Le projet **Hagimule** a permis de développer une infrastructure robuste et performante pour le téléchargement parallèle de fichiers volumineux.La mise en œuvre de composants clés comme le **Diary**, le **Downloader**, et le **Daemon** a permis d'assurer une coordination efficace entre les clients et les sources. La centralisation des métadonnées via une base de données a également simplifié la gestion des fichiers, tout en minimisant les calculs et en garantissant l'intégrité des données.

L'évaluation des performances a révélé que le téléchargement parallèle offre des gains considérables en termes de temps, surtout avec plusieurs sources. Cependant, pour des fichiers de petite taille, la compression n'apporte pas toujours des avantages nets en raison du coût de traitement associé. En revanche, pour des fichiers volumineux, comme ceux de plusieurs centaines de gigaoctets, la compression s'est avérée être une stratégie clé pour réduire significativement les délais.