

Université de Corse Pasquale Paoli



Rapport projet TCP/IP
(Application Client / Serveur)

Réalisé par :

- Chaker YAAKOUB
- Yasmine BOUKHALFA

Enseignant :

PA.BISGAMBIGLIA

Table des matières :

1 Introduction	3
2 Techniques de compression.....	4
3 Codage Huffman	4
4 Implémentation	6
5 Interface & résultat.....	10
6 Conclusion.....	12
7 Références.....	13

1. Introduction :

Dans le monde interconnecté d'aujourd'hui, où la gestion des données revêt une importance cruciale, la compression de données joue un rôle essentiel dans les architectures client-serveur modernes. Cette technique est particulièrement pertinente dans les scénarios de communication par socket, où chaque octet transmis compte.

Notre projet consiste en le développement d'une application Client/Serveur dédiée à la récupération de fichiers depuis un serveur distant. Grâce à l'utilisation du protocole de communication TCP-IP, les clients auront la possibilité de récupérer des fichiers stockés sur le serveur de manière rapide et sécurisée. L'objectif principal de cette application est de faciliter le transfert de données entre les clients et le serveur, en minimisant les contraintes liées à la distance géographique et aux limitations de bande passante.

La compression des fichiers tient un rôle central dans ce projet. En compressant les fichiers avant leur transfert, notre ambition est d'optimiser l'utilisation de la bande passante et de réduire significativement le temps nécessaire pour les téléchargements. Cette approche permet non seulement d'accélérer le processus de transfert, mais également de réduire les coûts associés à la consommation de bande passante.

Quant aux fonctionnalités de l'application, elles sont conçues pour offrir une expérience utilisateur fluide et complète. Les clients pourront naviguer à travers la liste des fichiers disponibles sur le serveur, sélectionner ceux qu'ils souhaitent télécharger, et effectuer ces téléchargements de manière transparente. De plus, l'application propose la possibilité d'ajouter de nouveaux fichiers à compresser et de décompresser les fichiers téléchargés localement sur la machine cliente. Notre objectif est de fournir une solution intuitive et efficace pour la gestion des fichiers à distance, répondant ainsi aux besoins variés des utilisateurs dans notre environnement numérique en constante évolution.

2. Techniques de compression :

Dans le domaine de la compression de données, plusieurs techniques sont largement utilisées pour réduire la taille des fichiers sans perte significative d'informations. Parmi les techniques les plus courantes, on trouve :

- **Compression sans perte :**

Cette technique vise à réduire la taille des données sans aucune perte d'information. Parmi les algorithmes les plus connus, on trouve :

- ✓ L'algorithme de Huffman : Il attribue des codes de longueur variable aux symboles en fonction de leur fréquence d'apparition dans les données.
- ✓ L'algorithme de Lempel-Ziv (LZ77, LZ78) : Il recherche les motifs répétitifs dans les données et les remplace par des références à des occurrences précédentes.

- **Compression avec perte :**

Cette technique permet de réduire la taille des données en acceptant une certaine perte d'information. Parmi les techniques les plus utilisées, on trouve :

- ✓ La transformation en ondelettes : Elle décompose les données en différentes échelles et résolutions pour représenter les détails avec une précision variable.
- ✓ La quantification vectorielle : Elle réduit la précision des données en regroupant les valeurs similaires dans des clusters.

- **Compression hybride :**

Cette approche combine des techniques de compression sans perte et avec perte pour obtenir un meilleur compromis entre la taille des données et la qualité de la reconstruction.

Notre projet repose sur la technique de compression sans perte, utilisant l'algorithme de Huffman.

3. Codage Huffman :

1. Principe :

Le principe du codage de Huffman repose sur la création d'un arbre composé de nœuds. On recherche tout d'abord le nombre d'occurrences de chaque caractère. Chaque caractère constitue une des feuilles de l'arbre à laquelle on associe un poids valant son nombre d'occurrences. Puis l'arbre est créé suivant un principe simple : on associe à chaque fois les deux nœuds de plus faibles poids pour donner un nœud dont le poids équivaut à la somme des poids de ses fils jusqu'à n'en avoir plus qu'un, la racine. On associe ensuite par exemple le code 0 (ou false) à la branche de gauche et le code 1 (ou true) à la branche de droite.

Pour obtenir le code binaire de chaque caractère, on remonte l'arbre à partir de la racine jusqu'aux feuilles en ajoutant à chaque fois au code un 0 ou un 1 selon la branche suivie.

Ci-dessous se trouve l'algorithme de Huffman :

```
Fonction Huffman → arbre
  données : l'alphabet et les fréquences de chaque lettre
  résultat : l'arbre binaire du codage de Huffman
  Algorithme
    créer une file à priorités
    pour chaque caractère c de l'alphabet faire
      créer un arbre a
      a.caractère ← c
      a.poids ← fréquence de c
      ajouter a à la file à priorités
    tant que la file à priorités contient plus d'un arbre faire
      créer un arbre a
      a.gauche ← l'arbre de poids le plus faible de la file à priorités (que l'on retire de la file)
      a.droite ← l'arbre de poids le plus faible de la file à priorités (que l'on retire de la file)
      a.poids ← a.gauche.poids + a.droite.poids
      ajouter a à la file à priorités
    retourner le seul arbre contenu dans la file à priorités
```

Figure 01 : Algorithme de Huffman

2. **Création de l'arbre** : L'algorithme de Huffman commence par analyser les fréquences d'apparition de chaque symbole dans les données à compresser. Ensuite, il construit un arbre binaire où les feuilles représentent les symboles et les chemins de la racine aux feuilles représentent les codes binaires attribués à chaque symbole.
3. **Attribution des codes** : Les codes binaires sont attribués aux symboles en parcourant l'arbre de Huffman. À chaque division dans l'arbre, un bit est ajouté au code du symbole correspondant. Les symboles les plus fréquents sont placés près de la racine de l'arbre, ce qui permet d'utiliser des codes plus courts pour les symboles les plus courants, contribuant ainsi à une compression plus efficace.
4. **Compression du fichier** : Une fois que l'arbre de Huffman est construit et que chaque caractère est associé à un code binaire, nous entamons le processus de compression. Nous parcourons à nouveau le fichier d'entrée et substituons chaque caractère par son code binaire correspondant.

5. **Décompression** : Pour décompresser les données, le fichier compressé est parcouru à partir du début. À chaque séquence de bits, l'arbre de Huffman est utilisé pour retrouver le symbole correspondant. Ce processus est répété jusqu'à ce que tous les symboles aient été décompressés et que les données originales soient récupérées

4. Implémentation :

1. Structure du projet :

Notre projet est structuré comme suit :

- **server.py** : Script Python pour l'implémentation du serveur.
- **client.py** : Script Python pour l'implémentation du client.
- **./huffmanCode/huffman.py** : Script Python contenant l'implémentation de l'algorithme de codage Huffman.
- **index.html** : Fichier HTML pour l'interface client.
- **DataBase** : Dossier contenant les scripts pour créer la base de données.
- **files** : Dossier contenant tous les fichiers compressés '.bin'.

2. Implémentation de codage Huffman

La classe **HuffmanCoding** implémente les fonctionnalités de compression et de décompression utilisant l'algorithme de codage Huffman. Voici un résumé des fonctionnalités principales :

Compression :

- **make_frequency_dict(text)**: Cette méthode calcule la fréquence d'apparition de chaque caractère dans le texte donné et retourne un dictionnaire contenant ces fréquences.
- **make_heap(frequency)**: Crée un tas de nœuds à partir des fréquences des caractères.
- **merge_nodes()**: Fusionne les nœuds du tas pour former l'arbre de Huffman en utilisant l'algorithme de fusion des deux nœuds de fréquences minimales.
- **make_codes_helper()**: Cette méthode récursive parcourt l'arbre de Huffman pour attribuer des codes binaires à chaque caractère.

- `get_encoded_text(text)`: Remplace chaque caractère du texte par son code binaire correspondant à l'aide de la table de codage créée précédemment.
- `pad_encoded_text(encoded_text)`: Ajoute un remplissage au texte encodé pour que sa longueur soit un multiple de 8, nécessaire pour l'encodage en octets.
- `get_byte_array()`: Convertit le texte encodé en un tableau d'octets pour le stockage et la transmission efficaces.

Décompression :

- `remove_padding(padded_encoded_text)`: Supprime le remplissage ajouté pendant la compression pour retrouver le texte encodé original.
- `decode_text(encoded_text, reverse_mapping)`: Décode le texte encodé en utilisant la table de mapping inverse, qui associe les codes binaires aux caractères correspondants.

Méthodes principales :

- **`compress(content)`**: Cette méthode prend en entrée le contenu à compresser, utilise les méthodes de compression décrites ci-dessus, et retourne le contenu compressé ainsi que la table de mapping inverse.
- **`decompress(content, reverse_mapping)`**: Prend le contenu compressé et la table de mapping inverse en entrée, et décompresse le contenu original en utilisant les méthodes de décompression.

3. Application client / serveur :

Dans une architecture de communication client-serveur utilisant TCP/IP, le serveur reste en attente des connexions des clients et les assiste en répondant à leurs requêtes. Un aspect essentiel de ce processus est la transmission de données compressées, qui vise à réduire la taille des données échangées sur le réseau, améliorant ainsi l'efficacité de la communication.

Voici le déroulement de l'échange, en mettant particulièrement l'accent sur le transfert de données compressées :

1. Etablissement de la connexion :

Dans la première étape, appelée l'établissement de la connexion, le serveur initialise un socket et se met à l'écoute sur une adresse IP et un port défini. De son côté, le client crée également un socket et s'y connecte en utilisant l'adresse IP et le port du serveur. Une fois que cette connexion est établie entre le client et le serveur, ils peuvent alors procéder à l'échange de données.

2. Echange de données :

Dans la deuxième étape, appelée l'échange de données, le client envoie une demande au serveur, comme par exemple demander un fichier spécifique. Le serveur reçoit cette demande, la traite et envoie une réponse appropriée. Par exemple, si le client demande un fichier, le serveur peut lui envoyer le contenu de ce fichier. Si les données à transférer sont importantes en taille, il peut être utile de les compresser pour économiser de la bande passante. Cela permet d'améliorer les performances du transfert et de réduire l'utilisation du réseau. Pendant cet échange de données, le serveur et le client utilisent les fonctions `send()` et `recv()` pour envoyer et recevoir des informations respectivement. Avant l'envoi, les données sont converties en octets avec la méthode `encode()`, et à la réception, elles sont décodées en chaînes de caractères avec la méthode `decode()`.

3. Compression des données :

Le serveur utilise un algorithme de compression tel que Huffman pour réduire la taille des données avant de les envoyer. Cette compression permet de diminuer la quantité de données transmises sur le réseau, ce qui peut accélérer le transfert et économiser de la bande passante.

4. Décompression des données :

Lorsque les données compressées parviennent au client, il les décompresse à l'aide du même algorithme utilisé par le serveur. Cela permet de restaurer les données à leur état original, les rendant utilisables selon les besoins du client.

Si nous souhaitons approfondir davantage la communication entre le client et le serveur dans le cadre de ce projet, voici un résumé des fonctionnalités des deux côtés :

1. Côté Client :

Technologies utilisées (FastAPI, HTML) :

- L'interface client est développée avec FastAPI pour l'API web et HTML pour l'interface utilisateur conviviale.

Processus de compression et de décompression des fichiers :

- Lorsqu'un fichier est envoyé au serveur, il est compressé avec l'algorithme de Huffman.
- Avant de télécharger un fichier depuis le serveur, le client reçoit les informations du fichier compressé ainsi que les données de mapping, puis effectue la décompression localement.

Fonctionnalités de téléchargement et d'envoi de fichiers :

- Le client peut envoyer un fichier au serveur pour l'enregistrement via une requête POST.
- Pour télécharger un fichier, le client envoie une demande avec le nom du fichier souhaité et le serveur répond en envoyant le contenu du fichier compressé avec son mapping Huffman.

2. Côté Serveur :

Gestion des connexions clients simultanées avec des threads :

- Le serveur utilise des threads pour gérer plusieurs connexions clientes simultanées, assurant un traitement efficace des requêtes concurrentes.

Réception, compression et stockage des fichiers :

- Lorsqu'un fichier est reçu, le serveur le compresse avec l'algorithme de Huffman avant de le stocker sur le disque.
- Chaque fichier est enregistré dans un répertoire dédié pour un accès organisé.

Utilisation d'une base de données pour stocker les informations sur les fichiers :

- Les détails des fichiers, tels que le nom original, le chemin d'accès sur le serveur et le mapping Huffman, sont stockés dans une base de données SQLite.
- Cette base de données permet une gestion efficace des fichiers et facilite les opérations de recherche et de récupération côté serveur.

5. Interface & résultats :

Voici l'interface de notre application, permettant l'ajout de nouveaux fichiers à compresser et le téléchargement des fichiers compressés :

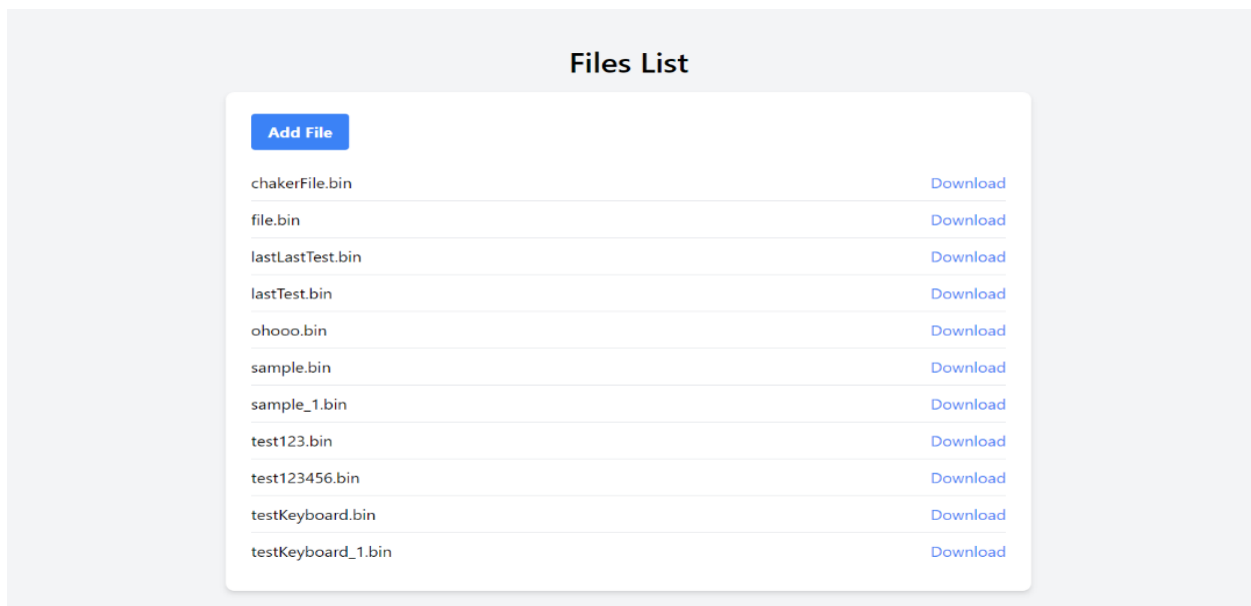
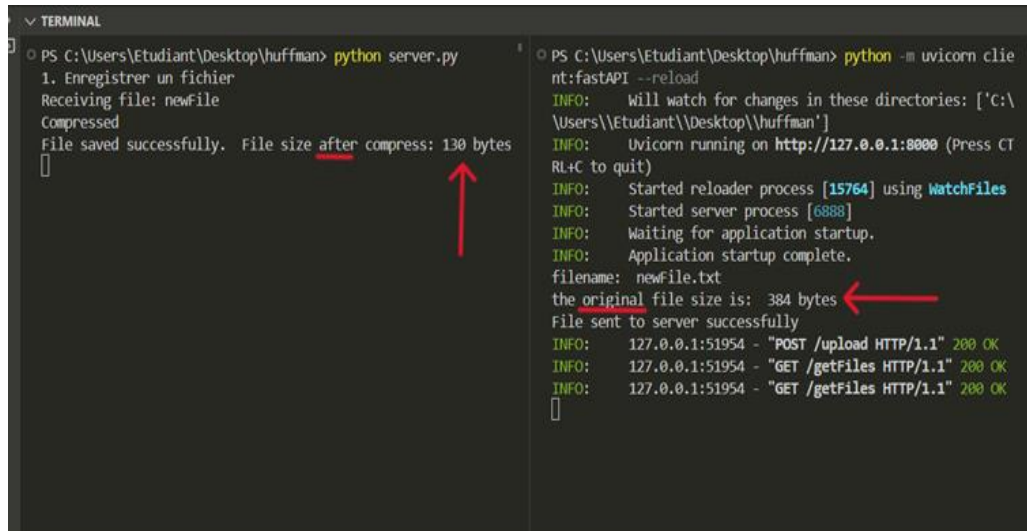


Figure 02: Interface utilisateur - index.html

1. Ajouter un fichier :

Pour ajouter un nouveau fichier, le compresser sur le serveur, puis le sauvegarder dans **file.bin**. Si un fichier portant le même nom existe déjà, un compteur est ajouté pour garantir l'unicité des noms de fichiers.



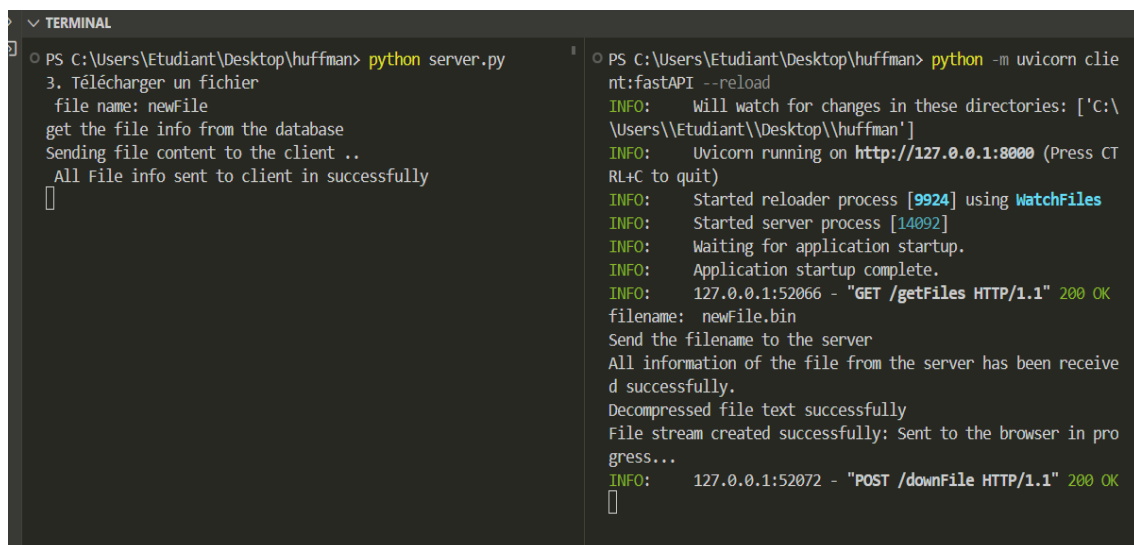
```
PS C:\Users\Etudiant\Desktop\huffman> python server.py
1. Enregistrer un fichier
Receiving file: newFile
Compressed
File saved successfully. File size after compress: 130 bytes

PS C:\Users\Etudiant\Desktop\huffman> python -m uvicorn client:fastAPI --reload
INFO: Will watch for changes in these directories: ['C:\Users\Etudiant\Desktop\huffman']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [15764] using WatchFiles
INFO: Started server process [6888]
INFO: Waiting for application startup.
INFO: Application startup complete.
filename: newfile.txt
the original file size is: 384 bytes
File sent to server successfully
INFO: 127.0.0.1:51954 - "POST /upload HTTP/1.1" 200 OK
INFO: 127.0.0.1:51954 - "GET /getFiles HTTP/1.1" 200 OK
INFO: 127.0.0.1:51954 - "GET /getFiles HTTP/1.1" 200 OK
```

Figure 03 : Affichage du terminal lors de l'ajout d'un nouveau fichier pour le compresser

2. Après le téléchargement :

Nous récupérons les données de la base de données telles que le mapping inversé et le contenu compressé du serveur. Le serveur envoie ensuite ces données au client, qui les décompresse et renvoie le fichier en tant que réponse en streaming vers le navigateur du client.



```
PS C:\Users\Etudiant\Desktop\huffman> python server.py
3. Télécharger un fichier
file name: newFile
get the file info from the database
Sending file content to the client ..
All File info sent to client in successfully

PS C:\Users\Etudiant\Desktop\huffman> python -m uvicorn client:fastAPI --reload
INFO: Will watch for changes in these directories: ['C:\Users\Etudiant\Desktop\huffman']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [9924] using WatchFiles
INFO: Started server process [14092]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:52066 - "GET /getFiles HTTP/1.1" 200 OK
filename: newFile.bin
Send the filename to the server
All information of the file from the server has been received successfully.
Decompressed file text successfully
File stream created successfully: Sent to the browser in progress...
INFO: 127.0.0.1:52072 - "POST /downFile HTTP/1.1" 200 OK
```

Figure 04 : Affichage du terminal après le téléchargement d'un fichier

6. Conclusion :

En conclusion, ce projet offre une solution complète pour la compression et le transfert de fichiers dans un environnement client-serveur. En utilisant des techniques de compression comme Huffman et en mettant en œuvre des fonctionnalités telles que l'ajout, le téléchargement, et la compression des fichiers, cette application offre une expérience utilisateur efficace et conviviale. Grâce à la combinaison de ces fonctionnalités, les utilisateurs peuvent transférer des fichiers de manière rapide et sécurisée, tout en optimisant l'utilisation de la bande passante du réseau. En outre, la décompression côté client assure une expérience transparente, permettant aux utilisateurs de récupérer facilement leurs fichiers dans leur format d'origine.

En résumé, ce projet démontre l'efficacité et la praticité de l'utilisation de la compression des données dans le contexte d'une application client-serveur.

7. Références

Liste des ressources utilisées pour le développement de l'application :

- Documentation FastAPI : <https://fastapi.tiangolo.com/>
- Documentation SQLite : <https://www.sqlite.org/docs.html>
- Documentation de Python sockets: <https://docs.python.org/3/library/socket.html>
- Documentation de l'algorithme de Huffman : [codage-de-huffman-tout-savoir](#)

Github link :

<https://github.com/ChakerYaakoub/huffman-coding->