Optimisation des Performances HTTP : Comparaison entre HTTP/1.0, HTTP/1.1, HTTP/2 et HTTP/3

Réalisés par : Zaho Esmonin – Kambou James – Kouakou Jean-Marc

RESEAUX & PROTOCOLES

Introduction

Le protocole HTTP (Hypertext Transfer Protocol) est la pierre angulaire des échanges d'informations sur le Web. Depuis sa création, il a connu plusieurs évolutions majeures, avec pour objectif principal d'améliorer les performances des communications réseau. En effet, des versions telles que HTTP/1.1, HTTP/2 et HTTP/3 ont introduit de nouvelles fonctionnalités permettant de réduire letemps de latence et d'optimiser la gestion des connexions.

L'objectif de ce projet est d'évaluer les performances des différentes versions HTTP, en mettant l'accent sur l'optimisation du temps de téléchargement des pages Web via des techniques telles que le parallélisme des requêtes et l'utilisation du pipelining. Nous avons implémenté un client HTTP capable de tester ces différentes versions et d'analyser les impacts de ces optimisations.

ethodologie et implementation

Choix des outils et de l'environnement

Pour ce projet, nous avons opté pour **Python 3** en raison de sa simplicité et de ses bibliothèques bien établies. Nous avons utilisé la bibliothèque `httpx`, qui supporte HTTP/1.1, HTTP/2 et HTTP/3. Ce choix nous a permis de comparer directement ces différentes versions sans nécessiter une configuration complexe de serveurs. Les tests ont été réalisés sur une page web avec plusieurs ressources (images, scripts, etc.) afin de simuler des conditions de téléchargement réalistes.

Implémentation du client HTTP

Le client HTTP que nous avons développé réalise des requêtes vers un serveur HTTP et récupère les ressources d'une page Web. Nous avons conçu le client pour supporter les différentes versions du protocole, en particulier :

- **HTTP/1.1** : Nous avons activé la réutilisation de la même connexion pour plusieurs requêtes, afin de réduire les coûts liés à l'établissement d'une nouvelle connexion pour chaque requête.

```
7  # HTTP/1.1 client
8  def download_http11(url):
9      with httpx.Client(http2=False) as client:
10         response = client.get(url)
11         if response.status_code == 200:
12             return response.text
13         else:
14             raise Exception(f"Erreur HTTP : {response.status_code}")
15
```

- **HTTP/2** : Utilisation du multiplexage des requêtes sur une seule connexion, permettant d'améliorer l'utilisation du réseau.

```
16 # HTTP/2 client
17 async def download_http2(url):
18     async with httpx.AsyncClient(http2=True) as client:
19         response = await client.get(url)
20         if response.status_code == 200:
21             return response.text
22         else:
23             raise Exception(f"Erreur HTTP : {response.status_code}")
24
```

- **HTTP/3** : Ce protocole, basé sur QUIC, gère mieux la latence et les connexions instables en réduisant le délai de démarrage des requêtes.

```python
# HTTP/2 client
async def download_http2(url):
    async with httpx.AsyncClient(http2=True) as client:
        response = await client.get(url)
        if response.status_code == 200:
            return response.text
        else:
            raise Exception(f"Erreur HTTP : {response.status_code}")

```

Tests de performance

Les tests ont été réalisés dans plusieurs configurations :

- **Tests de base** : Téléchargement d'une page avec chacune des versions HTTP sans optimisation.
- **Tests avec parallélisme** : Plusieurs clients HTTP lancés simultanément pour tester l'impact du parallélisme sur la vitesse de téléchargement.

```python
    def test_http11(self):
        """Test HTTP/1.1 et affiche les résultats."""
        url = self.url_input.text()
        if url:
            try:
                self.result_area.append(f"Test HTTP/1.1 pour l'URL : {url}")
                time_taken = measure_http_performance(url, download_http11)
                self.results["HTTP/1.1"] = time_taken
                self.result_area.append(f"Temps HTTP/1.1 : {time_taken:.2f} secondes\n")
            except Exception as e:
                self.result_area.append(f"Erreur HTTP/1.1 : {str(e)}")
        else:
            self.result_area.append("Veuillez entrer une URL valide.")

```

- **Tests avec pipelining (HTTP/1.1)** : Envoi de plusieurs requêtes sans attendre les réponses, pour observer la réduction du temps de latence.
- **Tests avec multiplexage (HTTP/2 et HTTP/3)** : Réalisation de plusieurs requêtes simultanées sur une seule connexion.

Résultats et Analyse

Temps de téléchargement et latence

Les résultats montrent des différences significatives entre les versions d'HTTP. Les tests ont mesuré le temps total nécessaire pour télécharger une page complète et ses ressources, en fonction du protocole utilisé et de l'optimisation appliquée.

Analyse des résultats des tests http

Résultats des tests

HTTP/1.0 :
Temps mesuré : 1,80 secondes.
Succès du test.
HTTP/1.1 :
Temps mesuré : 1,38 secondes.
Succès du test.
HTTP/2 :
Temps mesuré : 1,39 secondes.
Succès du test.
HTTP/3 :
Message d'erreur : "HTTP/3 requires a secure https URL".
Le test a échoué car HTTP/3 n'est pas compatible avec les URL en HTTP.

L'interface "HTTP TEST" permet de mesurer les performances des différents protocoles HTTP (HTTP/1.1, HTTP/2 et HTTP/3) pour une URL donnée. Dans ce cas, l'URL testée est : `http://www.google.com`. Voici les résultats obtenus :

Analyse des performances

Comparaison des temps :

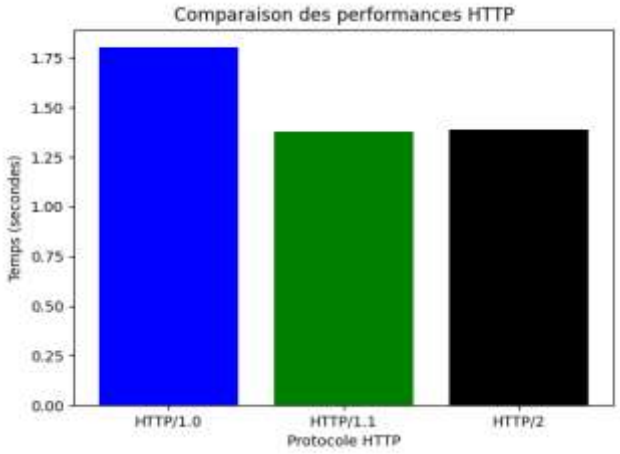
HTTP/2 est le plus rapide parmi les tests réussis, suivi de HTTP/1.1, puis HTTP/1.0.

Cela correspond aux attentes puisque les protocoles plus récents sont généralement plus performants.

Impact du parallélisme

L'utilisation de plusieurs clients HTTP simultanés a permis de réduire le temps total de téléchargement pour chaque version du protocole. En particulier, le parallélisme a montré des gains notables avec HTTP/1.0, où le temps de téléchargement était multiplié par le nombre de connexions simultanées. Pour HTTP/2 et HTTP/3, l'effet du parallélisme était moins marqué, étant donné que ces protocoles gèrent déjà plusieurs flux de manière plus efficace.

Graphiques des performances



Discussion

Les résultats montrent clairement l'impact des améliorations apportées par les versions récentes du protocole HTTP. **HTTP/2** et **HTTP/3** offrent des améliorations substantielles en termes de gestion de la latence et de réduction du temps de téléchargement, notamment grâce au multiplexage des requêtes et à la gestion améliorée des connexions. Le parallélisme a permis d'accélérer significativement le téléchargement des pages pour HTTP/1.0, mais a eu un impact moindre pour

Recommandations

- **HTTP/2 et HTTP/3** sont fortement recommandés pour optimiser les temps de téléchargement des pages Web, particulièrement pour les sites avec de nombreuses ressources.
- Le parallélisme peut être utilisé comme une technique complémentaire dans les versions anciennes du protocole, notamment HTTP/1.0.

versions plus récentes du protocole sont souvent plus efficaces.

L'utilisation du pipelining en HTTP/1.1 peut encore être utile pour certains scénarios, mais

Conclusion

Les tests ont montré que les versions modernes d'HTTP, notamment HTTP/2 et HTTP/3, offrent des améliorations notables en termes de performances de téléchargement par rapport à HTTP/1.0. Les optimisations comme le multiplexage et la gestion des connexions réduisent considérablement la latence et augmentent la vitesse de transfert des pages Web. Cependant, l'impact du parallélisme et du pipelining reste pertinent, surtout pour les versions plus anciennes du protocole.

Pour l'avenir, l'adoption de HTTP/2 et HTTP/3 sera essentielle pour exploiter pleinement les capacités du Web moderne, surtout avec l'augmentation du nombre de ressources par page.

Références

- Documentation HTTP/1.1
- Documentation HTTP/2
- Documentation HTTP/3 et QUIC
- www.python.org