Calcul Intensif: Distribution des Données & Calculs

Michée Allidjinou

January 8, 2024

Abstract

1 Problème

Le présent projet vise à résoudre le problème de téléchargement de gros fichiers dans un environnement distribué. Dans plusieurs applications, il est nécessaire de télécharger des fichiers depuis un serveurs distants puis de procéder à leur importation dans une base de données. Ce processus peut être automatisé dans ce que nous appellerons des batchs. Ces batchs s'exécutent de façon asynchrone. Il devient alors intéressant de pouvoir lancer un ou plusieurs téléchargement à la suite et de s'occuper d'autres tâches en attendant.

2 Solution

Pour résoudre le problème énoncé plus haut, nous avons décidé de développer un middleware en utilisant RabbitMQ. Le middleware se compose lui-même de deux parties: un backend classique, et un 'middleman'. Le backend n'est rien d'autre qu'un serveur HTTP qui reçoit des requêtes afin de lancer des tâches de téléchargement sur des workers. Le middleman quant à lui est chargé d'envoyer des notifications à un client grâce à une connection web-socket pour l'informer de l'état des différentes tâches lancées. Enfin, le worker est a pour seul but d'exécuter les tâches qui lui seront envoyées par le back-end.

3 Implémentation

La solution adoptée est développée en Go, un langage de programmation compilé, concurrent, et inspiré de C et Pascal développé par Google.

3.1 Architecture

L'architecture du système est présentée sur la figure suivante (Figure 1):

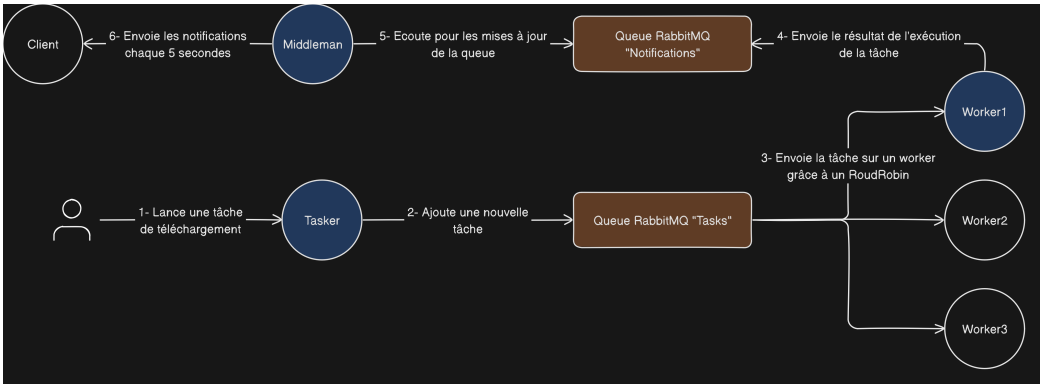

Figure 1: Architecture du système

Ce diagramme présente également les différentes étapes pour le téléchargement distant d'un fichier.

3.2 Les différents blocs de l'architecture

Comme mentionné plus haut, le système est composé de trois blocs principaux:

- Le backend, nommé ici "tasker"
- Le middleman
- Le worker

A ces blocs, nous pouvons ajouter RabbitMQ pour la communication asynchrone entre les différents éléments.

3.2.1 RabbitMQ

RabbitMQ est un logiciel d'agent de messages open source qui implémente le protocole Advanced Message Queuing [1]. Il est utilisé dans la communication dans le cadre d'un système distribué asynchrone.

Ici, nous faisons usage des queues de RabbitMQ et de son algorithme de RoundRobin pour distribuer les tâches entre les différents noeuds worker. Ainsi, à la réception d'une tâche par le backend, celui-ci publie un nouveau message sur la queue "tasks", celle-ci est une queue de travail (Work queue) à laquelle ont souscrit plusieurs noeuds worker. RabbitMQ envoie donc le nouveau message à un des noeuds selon son algorithme de RoundRobin. Nous avons donc un schéma de producteur-consommateurs où le client représente le producteurs (de tâches) au travers du backend et les noeuds works, les consommateurs.

3.2.2 Le backend

Le backend est un serveur HTTP écrit en Go. Il expose un endpoint "/task" qui supporte la méthode POST pour la création d'une nouvelle tâche. A la réception d'une nouvelle requête sur cet endpoint, le backend effectue des vérifications sur le corps de la requête pour s'assurer que celle-ci est valide. Dans notre cas, le corps doit être un objet JSON qui contient les clés suivantes:

- name: le nom de tâche à effectuer, permet de retrouver la tâche dans les notifications.
- arg: les arguments requis pour l'exécution de la tâche.
- taskType: le type de la tâche à exécuter. Pour le moment, nous ne prenons en compte qu'un seul type "download" qui permet de télécharge des ressources. Pour ce type, la clé "arg" contient l'url où se trouvent les ressources à télécharger.

Exemple du corps d'une requête pour le téléchargement de fichiers:

```
{
    "name": "download rabbitmq wikipedia page",
    "arg": "https://fr.wikipedia.org/wiki/RabbitMQ",
    "taskType": "download"
}
```

3.2.3 Le middleman

Le middleman a été développé pour permettre un monitoring des tâches en semi-temps réel. Pour cela, il écoute en permanence sur une queue RabbitMQ "notifications" et enregistre les nouveaux messages localement. Un client peut ensuite établir une connection websocket et alors le middleman envoie chaque 5 secondes les messages stockés. La mise en place d'un intervalle de temps permet de ne pas faire tourner une boucle infinie sur le middleman et ainsi économiser en ressource.

3.2.4 Le worker

Le worker est un programme également écrit en Go. Celui-ci reçoit les tâches à exécuter de la part de la queue de travail "tasks" et une fois la tâche exécutée, envoie le résultat de l'exécution sur une autre queue "notifications". Il est possible de lancer autant de worker que l'on souhaite.

References

[1] Wikipedia. *RabbitMQ*. 2022. URL: `https://fr.wikipedia.org/wiki/RabbitMQ` (visited on 01/08/2024).