TD 4 - ASR5 système d'exploitation

Protocoles et droits

29 avril 2018

I Protocole Pair à Pair

I.1 Contexte du travail

Les logiciels pair-à-pair ont connu un réel essort autour des années 2000, période où la recherche portait un intérêt tout particulier aux "overlay networks", c-à-d à un maillage logique de réseaux plus ou moins indépendant de la couche physique, des routeurs, ou des routeurs reconfigurables. Ils ont de multiples emplois, comme servir à échanger des fichiers ou communiquer par *chat*, et proposent des fonctionnalités qui leur sont propres et qui sont dépendantes de leur organisation : sécurité des communications, anonymat, authentification, etc.

Un pair est généralement volatile, et se définit comme égal en capacité de fonctionnement à tous les autres membres du réseau : il assure normalement les mêmes fonctions que les autres, à la fois client et serveur, quelque soit ses capacités physiques réelles (puissance de calcul, capacité de stockage, ou bande passante disponible).

Le shéma d'un maillage pair-à-pair revêt différentes formes : les réseaux pair-à-pair peuvent être à diffusion, ou organisés. Dans ce cas, ils peuvent reposer sur une grille logique (Lattice) ou sur un anneau logique (via une table de hachage distribuée, comme Chord, Pastry, Kademlia, etc.). Les moyens mis en œuvre pour la communication sont donc adaptés et propres à chaque réseau. En revanche, on retrouve les primitives de recherche, d'envoi et de réception dans chaqu'un d'eux.

Alors que certains hébergeurs offrent des possibilités de stockage dont l'accès est direct (Dropbox, Owncloud ou Uploaded), des protocoles pair-à-pair sont encore très utilisés, et continuellement améliorés au point de pouvoir constituer des alternatives aux systèmes de fichiers distribués (BitSync, IPFS), ou même réaliser une distribution de calculs sur de larges plates-formes (cassage de clés, branch-and-boud, etc.).

Pour un résumé, lire par exemple
`https://interstices.info/jcms/c_8622/les-reseaux-de-pair-a-pair`.

I.2 Fonctionnement

Nous allons aborder les réseaux pair-à-pair en s'inspirant du protocole de diffusion Gnutella dans sa version 0.4. Chaque pair est rélié à un certain nombre d'autres pairs (ses voisins directs), eux-mêmes connectés à d'autres pairs.

Entrée dans le réseau
Lorsqu'un nouveau pair souhaite participer, il doit obtenir un nombre suffisant de voisins. Pour cela, il doit uniquement connaitre un premier pair. Il lui demande de fournir un ensemble de machines auquel il pourra se connecter. S'il n'en obtient pas un nombre suffisant, il pourra recommencer la même demande...

Supposons que les pairs doivent maintenir un nombre de voisins compris entre 2 (au minimum) et 5 (au maximum). Pour créer des voisinages, l'arrivant doit demander à un premier contact une liste de pairs et pour chaque élément de la liste il doit demander si ce dernier accepte un nouveau voisinage.

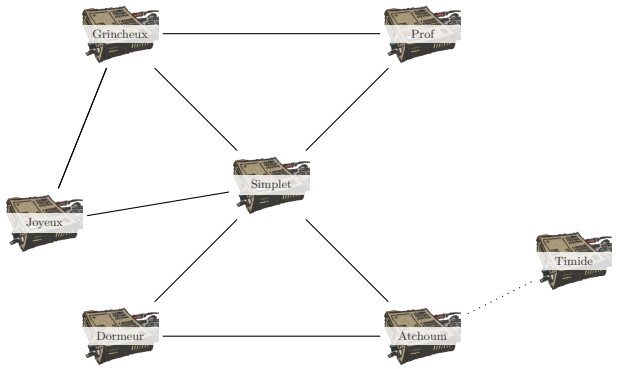

Figure 1 – Arrivée d'un pair

Q.I.1) - Proposer un algorithme de création des voisinages. En précisant les différents types de messages possibles, leur réponse ...

Solution: Il n'y a pas qu'une seule réponse à cette question. C'est a vous de définir un système cohérent. Ce qui suit n'est qu'un exemple.
Il faut définir le processus complet de création de voisinage. Le système n'est pas forcement symétrique, mais dans notre shema c'est e cas. Au minimum, il faut que le nouveau pair puisse se connecter aux autres et envoyer un premier message de demande de voisinage. Comme le nombre de voisins est limité, on est obligé de prévoir une possibilité de refu. Il faut donc qu'il existe :

- un message pour demander un voisinage,
- un message pour accepter un voisinage,
- un message pour refuser un voisinage.

Il faut aussi pouvoir demander au pair une liste d'autre pairs. Cela donne :

- un message pour demander une liste de pairs ;
- un ou plusieurs message(s) pour fournir la liste.

Il faut aussi définir les messages de manière à ce qu'ils soit lisibles. C'est à dire que le pair qui reçoit un message doit savoir ce qu'il a à lire et comment le lire. Par exemple, on peut décider que tout les messages sont des lignes de textes avec un fanion de fin (voir HTTP, SMTP). On peut définir des messages de taille fixe et connue par tous les pairs (mais cela rend plus difficile la transmission de listes). Enfin la forme la plus souple est celle qui propose un entête de taille fixe dans lequel il y a des informations sur la taille du reste du message.
Par exemple on peut proposer ici un système avec entête de taille fixe.
L'entête peut contenir :

- un numéro de version codé sur 2 chiffres décimaux ;
- l'adresse et le port de celui qui envoie le message (codé sur exactement 22 octets (XXX.XXX.XXX.XXX :PPPPPP) ;
- un type de messages (sur 4 octets) ;
- la taille des données du message sur 10 octets (des chiffres décimaux).

Chaque champs ici est de taille fixé, c'est très important pour permettre à celui qui lit l'entête de bien pouvoir le lire sans erreur. Il faut souvent faire attention à bien écrire les données. Par exemple, le pair d'adresse 10.250.11.1 port 22 enverra 010.250.011.001 :000022 et pas autre chose.
L'utilité de chaque champs est la suivante :

- le numéro de version est une sécurité, si à l'avenir on s'apperçoit que l'entête ne permet pas de coder un élément important, il suffit de créer une nouvelle version du protocole. Les lecteurs liront d'abord la version avant le lire la suite.

— l'adresse permet facilement d'ajouter le pair à la liste des voisins.
— le type de message permet de savoir ce que veut le pair :
  — `RqVo` requête pour demander un voisinage ;
  — `AcVo` accepte le voisinage ;
  — `NoVo` refuse le voisinage ;
  — `DePa` demande une liste de pairs ;
  — `LiPa` liste de pairs.
— la taille du message est nécessaire pour lire la suite sans problème.

Dans cet exemple, seul le dernier type à desoin de données, cela signifie que pour tous les messages, la taille sera `0000000000` sauf pour les message `LiPa` pour lequel on mettra la taille de la liste qui suit. La liste de pair envoyé peut être simplement une liste d'adresse de nom de machine avec les ports utilisés séparée par des « ; ».

Reste à définir le moment auquel on lance une connexion et celui où l'on coupe la communication. Par exemple, ici on peut décider que chaque connexion correspond à un seul échange (on pose une question et on recoit une réponse). Donc une connexion pour demander de créer un voisinage et recevoir son accord, une autre pour demander une liste de pairs et l'obtenir...

Plusieurs protocoles sont possibles, le moyen de vérifier si le protocole est correcte est de vérifier 2 choses :
— Est-ce que toute les informations qu'on veux tranmettre peuvent être représenté dans le protocole ?
— Est-ce qu'à tout moment chaque pair est capable de savoir qui va lire ou écrire et comment le lire ?

Q.I.2) - Quelles sont les informations échangées en suivant votre algorithme lorsque `Timide` arrive en contactant tout d'abord `Atchoum`.

Solution: On suppose que les IP des pair présent dans l'exemple sont bien celle utilisée et que le port est toujours 8080. Avec l'algo présenté à la question précédente :
— `Timide` se connecte à `Atchoum`
— `Timide`↦`Atchoum` : `01010.250.101.007:008080RqVo0000000000`
— `Atchoum`↦`Timide` : `01010.250.101.006:008080AcVo0000000000`
— Déconnexion
— `Timide` se connecte à `Atchoum`
— `Timide`↦`Atchoum` : `01010.250.101.007:008080DePa0000000000`
— `Atchoum`↦`Timide` :
  — `01010.250.101.006:008080LiPa0000000025`
  — `Simplet:8080;Atchoum:8080`
— Déconnexion
— `Timide` se connecte à `Simplet`
— `Timide`↦`Simplet` : `01010.250.101.007:008080RqVo0000000000`
— `Simplet`↦`Timide` : `01010.250.101.003:008080NoVo0000000000`
— Déconnexion
— `Timide` se connecte à `Dormeur`
— `Timide`↦`Dormeur` : `01010.250.101.007:008080RqVo0000000000`
— `Dormeur`↦`Timide` : `01010.250.101.005:008080AcVo0000000000`
— Déconnexion

Q.I.3) - Que manque-t-il pour maintenir un réseau de pair ?

Solution: Il faut considérer qu'un des pairs peut disparaitre. Le réseau doit alors se reconstituer. Pour cela, il faut que chaque pair envoie régulièrement une demande à ses voisins. Si le voisin ne répond pas, on l'élimine de la liste et on repasse en mode de recherche de voisins.

Par exemple, on crée 2 nouveaux types de paquets :

- `PING` demande d'une réponse
- `PONG` la réponse.

Régulièrement, le pair envoie un paquet `PING` à tout ses voisins et s'il n'arrive pas à se connecter ou si la réponse n'est pas un paquet valide de type `PONG`, le voisin est supprimé.

En général, on affecte un thread (voir le cours ASR7) à la tâche de maintenir la liste des voisins, c'est à dire de tester régulièrement tous les voisins, d'éliminer ceux qui ne répondent pas et d'en rechercher de nouveaux.