Emmanuel 24/06/2011

Guillot

RECOMMANDATION D'OBJETS PAR RELATIONS DE CONFIANCE DANS LES RESEAUX SOCIAUX

RAPPORT DE PFE DU DEPARTEMENT TELECOMMUNICATIONS, SERVICES & USAGES DE L'INSA DE LYON

Encadrants :

Frédérique Laforest	Maitre De Conférences HDR (Associate Professor)
	LIRIS, DRIM team
Simon Meyffret	Doctorant
	LIRIS, DRIM team

INTRODUCTION

Le PFE que j'ai effectué au LIRIS porte sur la recommandation d'objets par relations de confiance dans les réseaux sociaux. La recommandation d'objets est largement utilisée actuellement. Elle peut être sur des sites spécialisés dans la vente en ligne comme Amazon qui fait de la publicité ciblée par rapport aux pages que l'on a consultées. Cette recommandation peut aussi être, comme sur LastFm, sous la forme d'une « compatibilité » entre utilisateur. Si on aime certains artistes, on nous proposera d'autres artistes qui ont le plus été appréciés par les utilisateurs ayant aimé les même artistes que nous.

Ici, on ne se basera pas sur une similitude entre utilisateurs, mais sur un réseau social par relation de confiance : les utilisateurs choisissent les personnes à qui ils font confiance pour leur recommander des objets. Enfin, le choix qui a été fait est de protéger les informations des utilisateurs en utilisant des algorithmes distribués permettant aux utilisateurs de conserver leurs données uniquement pour les personnes qui leur font confiance.

Pour ce projet de recherche, j'ai travaillé durant la première partie de mon PFE sur la récupération d'un jeu de données sur le site Epinions, une référence dans le domaine de la recommandation. La deuxième partie de mon PFE s'est orientée sur l'implémentation des algorithmes de recommandation élaborés dans la thèse de Simon Meyffret sur une architecture pair à pair, via un simulateur p2p, pour voir les comportements de ces algorithmes sur une telle architecture.

TABLE DES MATIERES

ETAT DE L'ART

1. QU'EST CE QU'UN ASPIRATEUR DE SITE ?

Pour récupérer les informations nécessaires afin de construire le jeu de données, la première idée a été d'utiliser un aspirateur de site. Nous allons donc voir ici de quoi il s'agit.

Il s'agit d'un logiciel qui va permettre de parcourir et d'enregistrer toutes les pages d'un site web en local. On fournit à l'aspirateur de site l'adresse d'une page du site web et celui-ci va parcourir tous les liens internes pour récupérer l'ensemble de pages du site. Ce principe est utilisé par les robots des moteurs de recherche pour indexer les pages web.

Légalement, l'aspirateur de site doit se conformer aux informations contenues dans le fichier robots.txt qui se trouve à la racine du site et décrivant l'ensemble des pages autorisées et non autorisées.

Suite à de nombreuses recherches dans le domaine, j'ai sélectionné HTTrack Web Copier qui m'a semblé être un aspirateur de site pertinent et efficace.

2. RECUPERATION DES DONNEES

Nous allons maintenant voir les outils existant pour récupérer les données sur une page HTML.

La plupart des outils qui existent sont développés à la base pour des documents XML ou des documents HTML bien formés et non des documents HTML tels que ceux récupérés sur le site Epinions. Ainsi la librairie JDOM n'est pas adaptée à des pages en HTML mal formé. La première étape a été de trouver une librairie qui puisse « nettoyer » le code HTML. Les librairies TagSoup et HTMLCleaner rendent possible la récupération de documents HTML contenant de nombreuses erreurs.

TagSoup et HTMLCleaner prennent en entrée le code HTML et créent à partir de celui-ci une représentation DOM de la page permettant ainsi une navigation entre les différentes balises de la page HTML. Pour aller à des éléments précis, on peut aller de balise en balise en suivant une structure en arbre, une balise parente ayant plusieurs balises filles. Cette manière de procéder n'est pas toujours des plus pratiques. Il existe un langage créé pour les balises XML nommé XPATH qui permet d'atteindre rapidement les éléments recherchés. J'ai donc utilisé une API XPATH en java pour naviguer à travers la représentation DOM.

L'un des critères de choix du parseur HTML est donc de permettre d'utiliser XPATH. Après avoir testé XPATH avec TagSoup, je me suis rendu compte qu'en rectifiant le code HTML il ne permet plus d'utiliser XPATH correctement. Or HTMLCleaner incorpore une

implémentation du langage XPATH. Cette implémentation est certes très épurée mais tout de même suffisante dans notre cas. Les deux librairies étant équivalentes pour mes besoins sur les autres critères, j'ai choisi HTMLCleaner.

Malgré ces librairies très pratiques, j'ai tout de même dû utiliser des expressions régulières dans certains cas précis. En effet, la page HTML générée par Epinions ne contient aucun id et très peu de classes différentes, ce qui rend le parcours de la page à l'aide de XPATH assez compliqué. Dans certains cas une expression régulière s'est avérée beaucoup plus pratique.

Pour parser les pages d'Epinions j'ai donc utilisé une combinaison de nombreuses techniques existantes.

3. ENREGISTREMENT DES DONNEES

Au vu de la quantité de données à récupérer et à organiser, j'ai opté pour une base de données SQL. Le choix de la base de données s'est porté entre les deux bases de données open source MySQL et PostgreSQL. Après une étude des différences existant entre les deux bases de données, j'ai pu en conclure qu'elles possèdent à peu près les mêmes performances ainsi que les mêmes fonctionnalités. L'une des principales différences entre ces deux bases de données réside dans le fait que Mysql permet un traitement des duplicate key. Or dans les scripts de récupération des données, il est fort probable que nous trouvions plusieurs fois chaque utilisateur. Nous avons donc besoin de pouvoir les enregistrer en base en gérant le fait qu'un utilisateur existe déjà. C'est le critère qui m'a fait choisir MySQL plutôt que Postgresql.

4. PAIR A PAIR SOUS ANDROID

Dans cette partie, je vais faire un état de l'art des librairies open source existante afin d'avoir une idée de ce qui se fait dans le domaine.

SIP2PEER

Sip2Peer (s2p) est un middleware basé sur SIP pour l'implémentation d'applications pair à pair et d'applications distribuées. Sip2Peer est disponible pour JavaSE et Android et devrait bientôt être disponible sous iOS. La version 1.0 est disponible depuis le 12 mars 2011.

Sip2Peer fonctionne sur le principe d'un « BootStrap ». Au lancement d'un client, celui-ci se connecte à un « BootStrapping node » pour récupérer les informations nécessaires à la connexion au réseau Pair à Pair. Ce nœud renvoie au nouveau nœud la liste de ses pairs.

Afin de contrer les problèmes liés aux réseaux privés derrière un NAT, il existe en VoIp une solution qui est de déployer un Session Border Controller (SBC) afin de récupérer une adresse ip public. Sip2Peer étant basé sur SIP, la solution d'utiliser un SBC pour éviter les problèmes dus au NAT est donc naturellement implémenté.

Sip2Peer est distribué sous licence GPL version 2 ou plus. Cette licence oblige le code utilisant Sip2Peer à être lui aussi sous licence GPL.

ALLJOYN

Alljoyn est une technologie pair à pair permettant la communication mobile à mobile sans passer par un serveur intermédiaire. L'idée d'AllJoyn est de permettre de faire du pair à pair avec les mobiles voisins sans avoir à passer par internet. Si deux mobiles se trouvent à proximité, ils évitent ainsi de consommer leur connexion 3G pour communiquer mais passeront par le Wifi ou Bluetooth par exemple directement de mobile à mobile. Il permet de faire de l'authentification et du chiffrement sur les données transmises. AllJoyn est sous licence Apache 2.0.

Cette librairie est orientée vers une communication entre mobiles proches.

FROSTWIRE

Frostwire est un logiciel pair à pair open source multiplateforme développé en java. Il s'agit d'un logiciel et non d'une api. Il serait donc assez compliqué de réutiliser le code source de ce logiciel pour combler nos besoins.

Frostwire est sous licence GPL. Cette licence oblige le code utilisant Frostwire à être lui aussi sous licence GPL.

CONCLUSION PAIR A PAIR

Au regard des librairies détaillées ainsi que des autres librairies se rapprochant du sujet que je n'ai pas traité ici comme par exemple Smack (peu adapté pour le P2P au final), libjingle (pas encore disponible sur Android), on peut voir que Sip2Peer apparait comme la solution pour l'implémentation d'une application pair à pair simple sous android. Sip2Peer peut être implémenté en java, sous android et bientôt pour iOS. Il permet de créer des applications pair à pair simples avec si besoin la création d'un nœud annuaire et d'un SBC de façon très rapide. AllJoyn est quand à lui limité aux communications mobile à mobile ce qui ne nous intéresse pas, tout au moins pour le moment. Enfin Frostwire est un logiciel trop complexe pour être réellement utile dans notre cas.

Attention tout de même, Sip2Peer est sous licence GPL ce qui oblige le code l'utilisant à être lui aussi sous cette licence.

5. SIMULATION PAIR A PAIR

Après des recherches concernant les simulations de réseaux pair à pair, j'ai trouvé de nombreuses références au simulateur open source PeerSim. Ce simulateur permet de simuler les algorithmes pair à pair dont nous implémentons les protocoles. On peut ainsi tester l'efficacité de nos algorithmes assez simplement. PeerSim permet de simuler un très grand nombre de nœuds (plusieurs dizaines ou centaines de milliers de nœuds). De plus ce simulateur a déjà été utilisé par le LIRIS et a prouvé son efficacité. Ce simulateur permettra de tester les algorithmes efficacement.

6. REMARQUES

Sur la durée de mon PFE, je n'ai pas eu le temps d'implémenter le prototype Android. L'état de l'art sur cette partie sera néanmoins fortement utile lorsqu'il devra être implémenté. La partie Pair à Pair de mon PFE s'est limitée à la simulation de l'architecture à l'aide du logiciel PeerSim qui a constitué la deuxième partie de mon PFE.

RECUPERATION D'UN JEU DE DONNEES COMPLET

1. LE JEU DE DONNEES

EPINIONS

Nous allons voir maintenant la première partie de mon travail qui concerne la récupération d'un jeu de données complet sur le domaine de la recommandation par relation de confiance dans un réseau social. Pour récupérer ce jeu de données, nous avons décidé de nous intéresser au site Epinions. Plusieurs raisons nous ont conduits à ce choix.

Tout d'abord, Epinions est une référence dans le domaine de la recommandation ce qui nous permet d'avoir un jeu de données qui peut devenir une référence dans ce domaine. La deuxième raison est d'ailleurs directement reliée puisqu'il s'agit du fait que le précédent jeu de données provient lui aussi d'Epinions, mais date de plusieurs années et ne contient pas toutes les informations nécessaires à l'évaluation complète des algorithmes de la thèse de Simon Meyffret. Le précédent jeu de données était le seul disponible dans le domaine ce qui a eu pour effet qu'il soit pris comme référence dans de nombreux travaux.

LES FAIBLESSES DE L'ANCIEN JEU DE DONNEES

Comme nous l'avons vu juste avant, l'ancien jeu de données provient lui aussi du site Epinions. On peut donc se demander où est l'intérêt de récupérer un nouveau jeu de données provenant du même site. L'intérêt vient des limites des anciennes données. Celui-ci date en effet de plusieurs années ce qui le rend beaucoup moins complet que le site actuel.

De plus, il manque de nombreuses informations sur cet ancien jeu de données anonymisé. Les objets sont uniquement représentés par un identifiant de la même manière que les utilisateurs et les seuls autres informations présentes sont les notes des utilisateurs pour les objets ainsi que le réseau de confiance entre utilisateurs.

BUT D'UN NOUVEAU JEU DE DONNEES

Maintenant que l'on connait les faiblesses de l'ancien jeu, nous allons voir le but de la récupération d'un nouveau jeu de données. Ce nouveau jeu doit permettre de réutiliser les algorithmes existant et doit surtout permettre d'utiliser de nouvelles données dans ces algorithmes. Enfin, il doit être possible de le mettre à jour ou de rajouter facilement des données à celui-ci.

2. LES DONNEES

LOCALISATION DES DONNEES SUR LE SITE

Dans cette partie, nous allons voir la structure des données qui ont été récupérées. Sur la capture d'écran ci-dessous, voici à quoi ressemble la page d'un utilisateur actif d'Epinions.

Sur cette page, les éléments important sont :

1. 5 utilisateurs en qui « bigd99999 » a confiance. La liste de tous les utilisateurs en qui il a confiance est disponible sur une autre page.
2. 5 utilisateurs qui font confiance à « bigd99999 ». La liste complète se trouve aussi sur une autre page.
3. Les « expertises » d'un utilisateur. Il peut être top reviewer ou leader d'une ou plusieurs catégories (les catégories classent les objets).
4. Son rapport d'activité. On connait le nombre de reviews écrites par l'utilisateur.
5. Un objet noté par l'utilisateur.
6. La catégorie de l'objet noté.
7. La note donnée par l'utilisateur.
8. Ce que les autres utilisateurs ont pensé de la note donnée par « bigd99999 ».

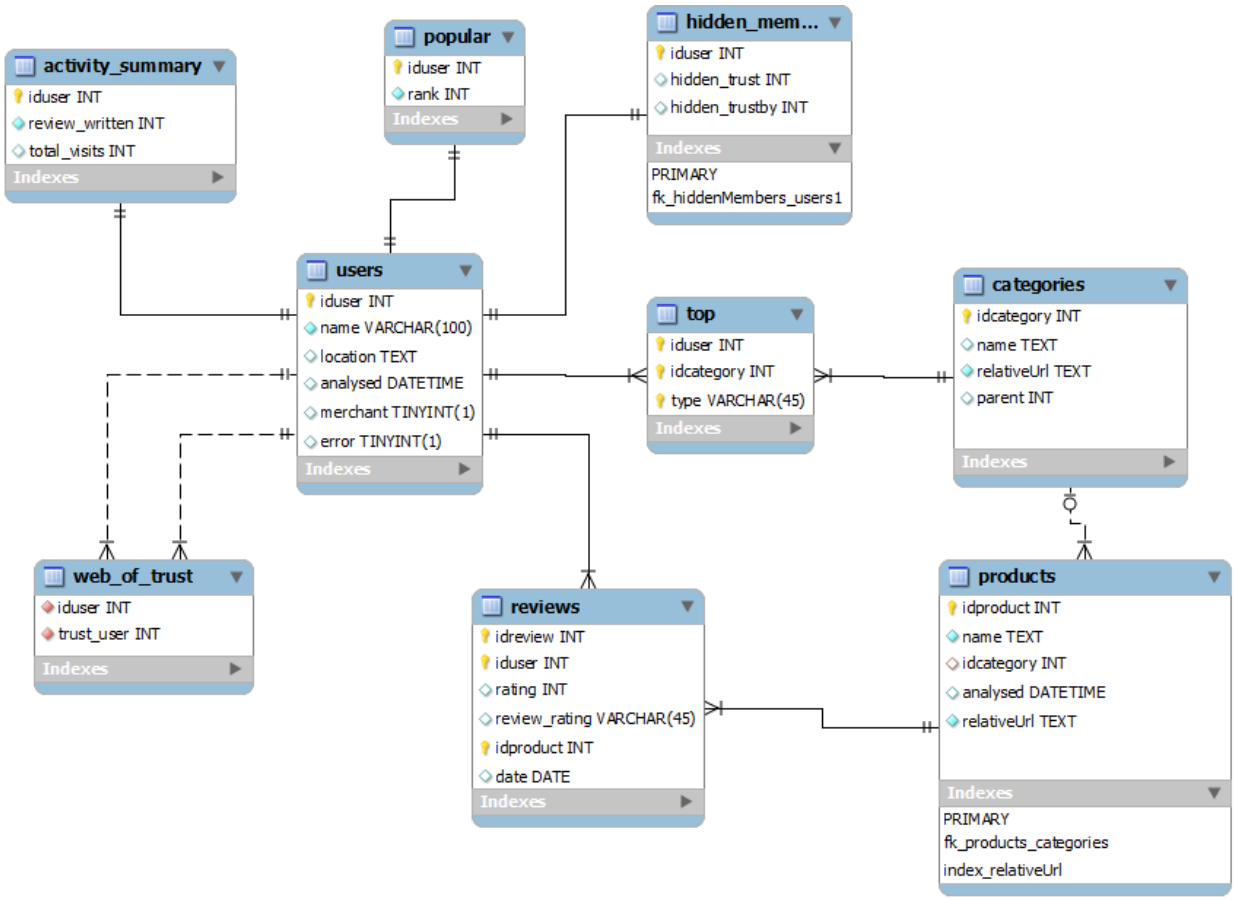
Comme on a pu le voir, l'ensemble des données n'est pas toujours regroupé sur la page de l'utilisateur. Pour des utilisateurs assez actifs comme « bigd99999 », il faut récupérer le reste du « Web of trust » et de ses notes sur d'autres pages.

Pour des utilisateurs ayant peu de reviews (5 ou moins) et autant de personnes dans leur « web of trust », il n'est pas nécessaire de parcourir d'autres pages.

Enfin, pour compléter le jeu de données, nous récupérons la catégorie des objets directement depuis la review d'un utilisateur. Cette catégorie est l'une des nouvelles informations de ce nouveau jeu de données. Les catégories sont organisées hiérarchiquement dans un arbre, les feuilles de cet arbre étant les objets. Pour rendre cette information aussi pertinente que possible, le script créé va récupérer également la catégorie parente de celle-ci si elle n'existe pas encore.

STRUCTURE DES DONNEES RECUPEREES

Dans cette partie, nous allons voir la structure des données stockées dans la base de données pour créer notre jeu de données. Voici un schéma de la base pour plus de clarté.



Nous avons vu la majeure partie de ces données précédemment. Un « User » possède un « Web of Trust », un ensemble de « reviews », et un « activity summary ». On stocke aussi les utilisateurs populaires et les « Top user» qui sont des membres très actifs sur Epinions et ayant une forte influence dans leur notation.

On peut aussi voir sur ce schéma les choix effectués sur l'enregistrement des données. Les utilisateurs et objets possèdent une date de dernière analyse. Cette date nous permet de savoir si l'utilisateur a bien été analysé et de quand l'analyse date. Il en va de même pour les objets bien que cette date n'ait pas été utilisée. En effet, nous récupérons les objets directement depuis les notes des utilisateurs sans analyser la page. J'ai pourtant opté pour l'enregistrement de l'URL des objets afin de les distinguer et de permettre de les analyser dans le cas où nous désirerions plus tard obtenir plus d'informations sur les objets.

On peut aussi observer pour les catégories que l'on stocke pour chacune d'elles son parent. Cette architecture d'héritage pourra être fort utile dans la recommandation.

Enfin, il reste dans la base certains éléments qui permettent de détecter des cas particuliers. Dans les utilisateurs, on stocke ceux qui sont en erreur pour pouvoir vérifier le

problème détecté, on stocke si un utilisateur est un marchand. Ils sont enregistrés de manière similaire à des utilisateurs sur Epinions mais ils n'ont pas le même rôle ni les même informations qu'un utilisateur normal. Ces marchands ne nous intéressent pas. Enfin, sur Epinions, les utilisateurs peuvent cacher les personnes à qui elles font confiance. J'ai voulu stocker cette information dans la table « hidden_members ».

PROBLEMES RENCONTRES ET CHOIX EFFECTUES

Comme nous l'avons vu dans l'état de l'art, j'ai effectué des recherches sur un aspirateur de site. Ce choix paraissait tout à fait indiqué étant donné la nature du travail à effectuer. Un aspirateur de site aurait permis de récupérer l'ensemble des pages du site Epinions et aurait permis d'avoir ainsi les données en local pour permettre de les traiter librement par la suite sans avoir à récupérer les pages depuis Epinions.

La première chose à faire avec un aspirateur de site pour faire ce genre de travail a été de filtrer les données qu'il devait récupérer et ne récupérer que les pages HTML. Le reste des données du site web ne nous intéresse pas.

Une fois la configuration effectuée, j'ai rencontré de nombreux problèmes avec l'extraction. Tout d'abord, sur les pages d'Epinions, il existe plusieurs « domaines » d'Epinions dont les trois exemples suivant:

- www.epinions.com
- www99.epinions.com
- www0.epinions.com

Du point de vue de l'aspirateur de site, il s'agit de domaines différents, or il a été créé à la base pour ne récupérer qu'un seul site. J'ai trouvé comment lui rajouter des noms de domaines, mais il n'y avait aucun moyen de savoir combien de noms de domaines différents existent pour Epinions.

Ensuite, le site d'Epinions est gigantesque et contient au final énormément de pages qui ne sont pas utiles pour la création de notre jeu de données. Comme nous le verrons dans les résultats, il a fallu plus de 40h pour récupérer et traiter l'ensemble des pages dont nous avions besoin. Récupérer l'ensemble des pages du site n'était donc pas envisageable.

Enfin dernier point, Epinions est un site très dynamique. Or l'aspirateur de site que j'ai utilisé ne peut être relancé là où il s'est arrêté lors d'une aspiration. Il vérifie automatiquement les pages déjà téléchargées pour vérifier si la date de validité et/ou la taille est différente de la version qu'il possède en local. Or sur Epinions, toutes les pages devaient être retéléchargées. L'impossibilité de stopper un téléchargement en cours posait un gros problème au vu du temps de récupération du site. Une erreur réseau au milieu et tout aurait été à refaire. Il doit surement exister certains aspirateurs de site permettant de reprendre un téléchargement là où ils se sont arrêtés, mais les autres arguments en

défaveur de l'aspirateur de site m'ont fait me diriger vers la création d'un téléchargement personnalisé.

En choisissant le téléchargement personnalisé, nous avons plus de liberté et de nombreux avantages. Nous pouvons récupérer les pages dans l'ordre désiré, pas de problème de configuration pour ne récupérer que le HTML, nous avons le libre choix pour télécharger uniquement les pages souhaitées. Nous avons aussi la possibilité d'arrêter le téléchargement à n'importe quel moment et de le relancer par la suite sans se soucier de vérifier les pages déjà téléchargées. Enfin, cela permet de faire des mises à jours partielles de la base de données et de ne pas récupérer les pages des objets tout en se laissant la possibilité de pouvoir les récupérer par la suite s'il faut plus d'informations sur une certaine catégorie d'objets pour l'algorithme de recommandation.

3. IMPLEMENTATION

Dans cette partie, nous allons voir tout ce qui concerne l'implémentation de la solution choisie pour l'extraction des données. Cette partie sera séparée en trois grandes étapes qui ont été celles de mon implémentation. Tout d'abord nous allons voir comment le choix de la stratégie de la récupération des données a été effectué, puis nous verrons la récupération des utilisateurs, et enfin la récupération des informations de ces utilisateurs.

STRATEGIE DE RECUPERATION DES DONNEES

Lorsqu'il a fallu au début déterminer une stratégie de récupération des données, la première idée était de récupérer l'ensemble des objets notés sur le site et par la même occasion de récupérer les utilisateurs ayant noté ces objets. Pour les récupérer, il existe sur le site Epinions un plan du site extrêmement détaillé avec quasiment toutes les catégories d'objets. Suivant ce constat, j'ai commencé à chercher des moyens de récupérer ces objets depuis ces catégories. Je me suis vite rendu compte que les pages contenant les listes d'objets étaient bien souvent différentes ce qui rendrait la récupération très compliquée.

Ensuite, une particularité d'Epinions est devenue fort problématique. Lorsque l'on est sur une page regroupant une liste d'objets, le nombre de résultat est limité à 100 pages, ce qui représente 1500 objets. Or de nombreuses catégories possèdent bien plus de 1500 objets. Il fallait donc trouver d'autres solutions.

L'analyse détaillée de la page des utilisateurs a montré qu'elle contenait toutes les informations dont nous avions besoin. Plus besoin d'aller récupérer les pages des objets. Mais comment récupérer la liste de ces utilisateurs ? C'est ce que nous allons voir dans la prochaine partie.

RECUPERATION DE L'ENSEMBLE DES UTILISATEURS

a) *Procédure de récupération*

Le site d'Epinions possède un moteur de recherche interne pour les utilisateurs. Celui-ci permet de retrouver normalement l'ensemble des utilisateurs. J'ai donc testé ce moteur de recherche avec la chaine « aa » et me suis rendu compte qu'il retrouvait des utilisateurs ayant « aa » dans leur nom. J'ai testé plusieurs combinaisons qui ont bien fonctionné. J'ai eu ainsi l'idée d'utiliser une méthode dictionnaire avec l'ensemble des combinaisons de deux lettres et/ou chiffres. En suivant la logique, cette méthode devrait permettre de récupérer l'ensemble des utilisateurs d'Epinions.

Grâce à cette méthode, nous avons pu récupérer pas moins de 240 000 utilisateurs. Pour récupérer ces utilisateurs, il a fallu 4h en comptant le téléchargement et l'analyse de toutes les pages. Pour améliorer les performances de mon script de récupération, j'ai mis en place du multithreading. J'ai limité le nombre de Thread à 4 pour conserver de bonnes performances tout en évitant de surcharger le site Epinions en lui envoyant un trop grand nombre de requêtes simultanées.

Le principe du script était très simple, on crée la liste des couples de lettre/chiffre et chaque thread prend l'un des couples pour récupérer les pages de résultats. Il récupère en même temps le nombre de pages de résultat pour ce couple de lettres et parcourt chacune de ces pages. Il récupère tous les liens sous la forme « /user-<Nom d'un User> » et stocke du coup le nom de cette utilisateur en base.

En utilisant cette méthode nous retrouvons très souvent plusieurs fois le même utilisateur. C'est ce qui a justifié l'utilisation de MySql plutôt que PostGreSql comme nous l'avons vu dans l'état de l'art. MySql gère l'insertion de clé identique à l'aide d'un « INSERT IGNORE ».

b) *Limites de cette récupération*

La récupération de l'ensemble des utilisateurs aurait du fonctionner à 100% si l'on en croit la logique. En effet, la méthode utilisée est exhaustive et n'aurait du laisser aucun utilisateurs de côté. Ce ne fut pas pour autant le cas.

Le problème de la limitation de 1500 résultats par recherche (les 100 pages de résultats) est aussi présent pour la recherche d'utilisateurs. Certains couples de lettres présentaient un problème sur ce point. Une idée a donc été de rajouter dans ces cas un triplet au lieu d'un couple de lettre. Or, pour certains utilisateurs, il s'agissait des deux dernières lettres ce qui les empêchaient d'être trouvés avec le triplet de lettre. De plus, pour l'un des triplets testés, il y avait aussi plus de 100 pages de résultats. J'ai donc abandonné l'idée de rajouter des triplets.

Un autre problème, plus grave a été de voir que certains couples de lettres ne comprenaient aucun résultat. Par exemple le couple « no » ne présente pas de résultat alors que le triplet « non » possède tout de même 4 pages de résultats.

Nous pouvons d'ores et déjà affirmer que cette méthode de récupération n'a pas été exhaustive. Ceci n'est pas si problématique qu'il n'y parait à première vue. En effet, chacun des 240 000 utilisateurs trouvés possède un « web of Trust ». Nous récupérerons les utilisateurs manquants dans celui-ci. Il restera bien sûr certains utilisateurs n'ayant aucun lien avec ce noyau des 240 000 utilisateurs, mais ceux-ci n'ont soit aucun lien et ne nous intéressent donc pas, soit ne sont pas connectés à notre jeu de données et donc n'influences pas les résultats. Nous ne cherchons pas non plus à être exhaustifs sur les données d'Epinions, mais à récupérer le plus grand jeu de données cohérent possible.

c) Améliorations possibles

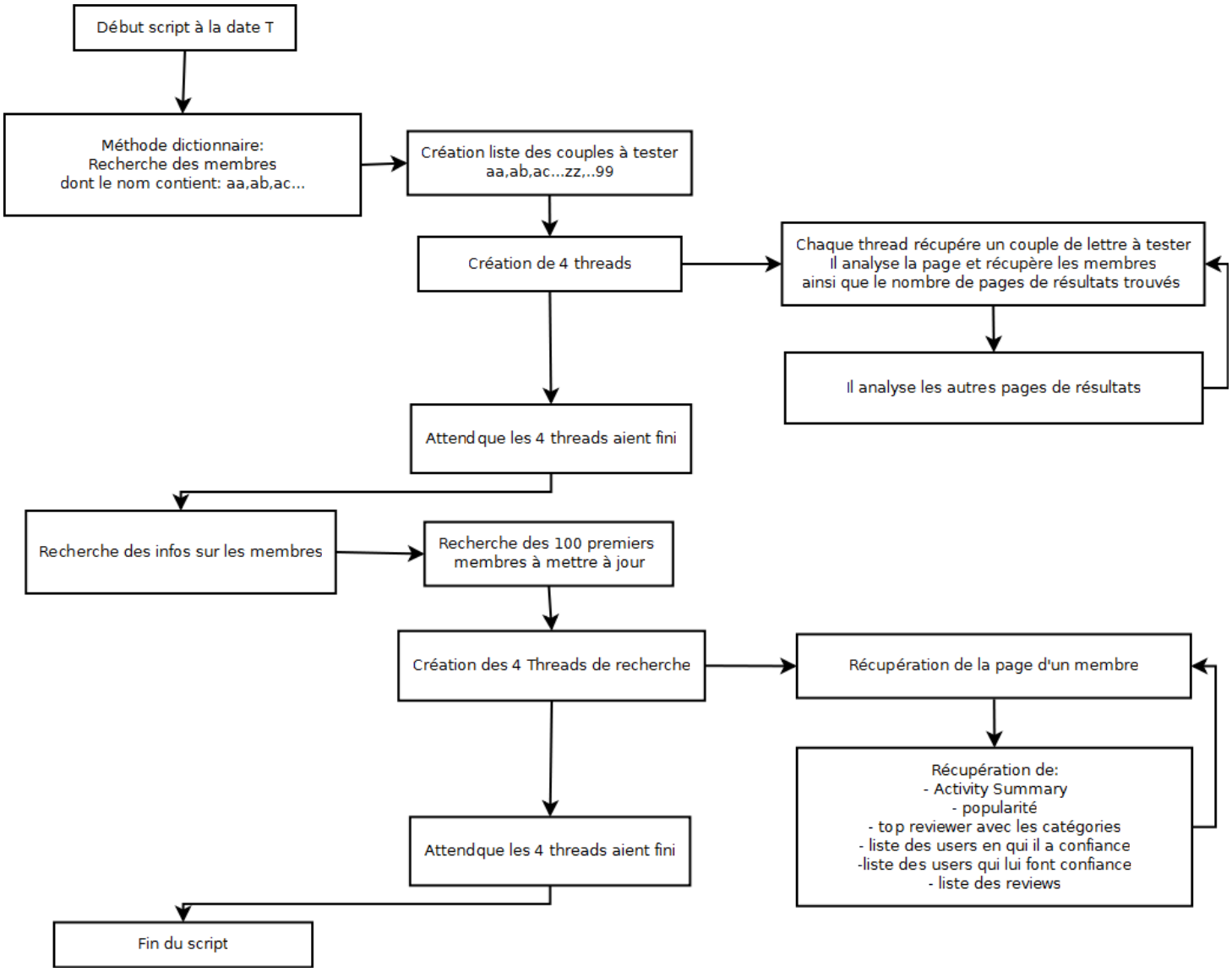
Les améliorations possibles pour la récupération d'utilisateurs sont variées. On peut mettre en place la solution des triplets de lettres dans le cas où trop d'utilisateurs ou aucun n'a été trouvé. Cette solution permettra surement de récupérer quelques utilisateurs supplémentaires. Une autre solution serait d'analyser l'ensemble des objets notés et de vérifier que l'ensemble des utilisateurs ayant noté ces objets est présent dans la base. Enfin, la dernière solution qui me parait être assez irréalisable serait d'effectuer des recherches directement depuis les catégories d'objets pour récupérer certains utilisateurs. Il s'agit de la solution que j'ai voulu éviter au début puisqu'elle possède plus d'inconvénients que d'avantages et utilisée seule, elle ne sera surement pas plus efficace que la solution implémentée.

Pour conclure, la solution actuelle n'est pas parfaite, elle est tout à fait appropriée dans notre cas puisqu'il ne s'agit pas d'être exhaustif. Et même si plusieurs milliers d'utilisateurs n'ont surement pas été trouvés, s'ils n'ont pas de réseau social, ils ne sont pas utiles dans le jeu de données.

RECUPERATION DES INFORMATIONS DE CHAQUE UTILISATEUR

a) Procédure de récupération

Maintenant que nous avons récupéré la majorité des utilisateurs, il faut s'occuper des informations de chacun d'eux. Le diagramme de séquence suivant présente l'algorithme utilisé.



La deuxième partie du diagramme correspond à la recherche des informations. Comme nous pouvons le voir, là aussi, nous avons du multithreading avec encore 4 threads pour conserver de bonnes performances sans surcharger le site Epinions. Nous avons dans la base, l'ensemble des utilisateurs (table users) récupérés qui ne sont pas analysés. Une requête SQL ira donc récupérer les 100 premiers users à analyser. Dès que la liste des users à analyser est vide, le script ira rechercher les 100 users suivant par une requête SQL et ainsi

de suite jusqu'à ce qu'il n'y ait plus aucun user dont la date d'analyse soit null ou antécédente à la date de lancement du script.

Avec cette méthode, on n'enregistre les informations d'un utilisateur qu'une fois que l'ensemble de ses informations a été récupéré. On est donc sûr de ne pas avoir des utilisateurs dans un état instable avec seulement une partie des informations récupérées. Si le script s'arrête pour une raison quelconque, seule l'analyse des 4 utilisateurs en cours de recherche sera perdue et lorsqu'il se relancera le script reprendra l'analyse par les 100 users les plus urgents à analyser. Il s'agira d'abords des utilisateurs n'ayant pas encore été analysés, puis de ceux dont la date d'analyse est la plus ancienne.

b) Optimisations

Au vu de la taille de l'analyse à effectuer, j'ai recherché à l'aide d'un profiler les méthodes à améliorer pour réduire au maximum le temps d'analyse. J'ai eu aussi quelques problèmes de mémoire qui se sont rapidement résolus en utilisant le profiler. Au-delà de l'optimisation de mes méthodes de calculs grâce au profiler et de l'ajout du multithreading, c'est dans la récupération des pages que mon script a dû être optimisé. J'ai donc cherché un moyen de réduire au maximum le nombre de pages visitées.

Sur la page d'un utilisateur, on peut ainsi récupérer jusqu'à 5 reviews, il n'est donc pas nécessaire d'aller sur la page dédiée aux reviews si dans l'activity summary de l'utilisateur il n'y a pas plus de 5 reviews.

Pour le web of trust, c'est le même principe. S'il y a plus de 5 membres, un lien sur la page de l'utilisateur amène vers l'ensemble des utilisateurs. Lorsque ce lien n'est pas présent, on ne récupère que les utilisateurs présents sur la page de l'utilisateur.

Une dernière optimisation a été d'enregistrer les pages analysées sur le disque dur pour avoir une version sur laquelle travailler en local dans le cas où une erreur dans le script de récupération aurait entrainé l'oubli de certaines informations. Seul les 80 000 derniers users ont été sauvegardés sous cette forme, mais ce sera toujours ça qui ne sera pas à retélécharger en cas de problème détecté.

c) Difficultés rencontrées

La récupération des informations a posé de nombreux problèmes dus à la structure des pages sur le site Epinions. En effet, les pages d'Epinions ne comprennent aucun Id pour distinguer les balises Html. Elles sont pleines de tableaux pour structurer le design du site, ce qui rend la récupération des éléments très compliquée.

Pour récupérer les informations, j'ai souvent utilisé XPath qui m'a permis de récupérer par exemple l'ensemble d'un tableau en récupérant chacune des lignes dans un tableau d'objets. Il était à partir de là possible de récupérer chacune des informations recherchées. Mais le manque d'Id dans la page HTML a rendu la recherche avec XPath beaucoup plus compliquée que prévu. Elle a aussi l'inconvénient qu'en cas de rajout

d'informations sur une page hors Epinions, l'expression ne sera peut être plus correcte. Si l'on dit à Xpath de récupérer le 4^{e} tableau de la page et qu'un autre tableau est inséré par Epinions plus tard, il faudra modifier les expressions pour que le script se retrouve fonctionnel à nouveau. Pour certaines informations, comme l'activity summary, une expression régulière s'est avérée plus pertinente que l'utilisation d'XPath.

Ensuite, les pages d'Epinions contiennent de nombreux cas particuliers que je n'ai pu traiter qu'une fois le script tombé sur celui-ci. Des utilisateurs récupérés dans le moteur de recherche contenaient des caractères spéciaux et n'avait du coup pas de page au final sur Epinions. Certains n'avaient pas de reviews, pas de Web of Trust ou encore avaient les membres de leur web of trust cachés.

Il y a encore d'autres particularités des pages Epinions comme par exemple dans les catégories. Sur les 580 catégories récupérées, 13 n'avaient pas le même « fil d'Ariane » qui permet d'en déduire la catégorie parente. La variante de leur fil d'Ariane a entrainé une erreur du script qui les indiquait comme parentes d'elles-mêmes.

Enfin, la dernière difficulté rencontrée a été pour le test des informations récupérées. Du fait de la grande variété des pages et de nombreux cas particuliers, une page type peut être récupérée sans problème dans les tests, mais un cas particulier nouveau va se retrouver en erreur et il faudra peut être l'analyse de 100 000 utilisateurs pour rencontrer ce cas là.

d) Améliorations possibles

Les améliorations possibles du script peuvent venir de plusieurs choses. La première vient des tests à effectuer. En effet, vu que les pages du site d'Epinions sont susceptibles d'évoluer, il serait préférable d'effectuer un test au lancement du script pour vérifier que la structure des pages n'a pas évolué entre temps.

Enfin, il reste de nombreuses informations supplémentaires à récupérer sur les utilisateurs qui pourraient s'avérer utile pour la recommandation. Ces informations ne sont pas nécessaires pour le moment, mais pourraient l'être dans l'avenir si l'algorithme de recommandation évolue. C'est pour cette raison que l'adresse des pages des objets notés est conservée en base pour permettre, au-delà de l'identification d'un objet, de visiter la page de chaque objet pour récupérer d'autres informations.

4. RESULTATS

QUANTITE DES DONNEES RECUPEREES ET PERFORMANCES

Nous allons maintenant voir les résultats obtenus dans la récupération du jeu de données. Comme dit précédemment, nous avions 240 000 utilisateurs récupérés à l'aide de la première méthode de récupération. Il a fallu 4h pour récupérer ces utilisateurs qui ont constitué le noyau d'analyse de la seconde partie.

Grace à la deuxième partie du script, nous avons pu compléter ces utilisateurs par tous les autres utilisateurs reliés à ceux-ci. Voici les principaux chiffres du nouveau jeu de données :

- 307 000 utilisateurs
- 261 utilisateurs « leader » d'une ou plusieurs catégories
- 318 000 objets
- 587 catégories
- 29 catégories « racines »
- 1 127 000 reviews/notes
- 538 700 liens de confiance

Ces chiffres montrent bien l'étendue du jeu de données récupéré. Pour ce qui concerne les 307 000 utilisateurs, il faut relativiser ce nombre car beaucoup d'entre eux n'ont pas d'informations. Il y a en tout 47 555 utilisateurs qui font confiance à au moins une personne et 55 186 ayant au moins une personne leur faisant confiance. Enfin il n'y a dans les reviews « que » 113 631 utilisateurs différents ayant participé à l'écriture d'au moins une review.

Au niveau des performances, il a fallu un peu plus de 40h pour récupérer ces informations avec un pc portable Intel Core 2 Duo GHz et 3Go de mémoire vive

INCOHERENCES DANS LA BASE DE DONNEES FINALE

Dans le jeu de données final, certaines incohérences ont été trouvées. Nous avions par exemple 13 catégories ayant elle-même pour parent. Comme expliqué auparavant, il s'agissait d'une particularité sur la page de ces 13 catégories qui n'a normalement pas lieu d'être. L'erreur a été corrigée.

Un autre problème détecté vient des reviews des utilisateurs. Certains utilisateurs ont noté plusieurs fois le même objet. Après vérification de certains des objets, il s'agit en réalité d' « objets » différents mais enregistrés sur Epinions comme un seul et même objet. On a par exemple des destinations qui sont répertoriés et chacune des reviews des utilisateurs fera référence à un monument, un musée,… à visiter pour cette destination. Il y a aussi des objets appartenant à une catégorie archive qui sont un regroupement d'anciennes reviews pour une même catégorie. Ces objets restent tout de même minoritaires.

SIMULATION PAIR A PAIR AVEC PEERSIM

1. INTRODUCTION

Nous passons maintenant à la deuxième partie de mon PFE basée sur la simulation pair à pair de la recommandation d'objets. Le système de recommandation doit pouvoir utiliser une architecture distribuée et fonctionner en pair à pair pour garantir la confidentialité des données des utilisateurs. Cette partie est donc primordiale pour faire la validation de l'algorithme de recommandation qui doit fonctionner sur le simulateur. La simulation à l'aide de PeerSim doit permettre de faire la validation des algorithmes sur une architecture distribuée.

2. FONCTIONNEMENT DE PEERSIM

PeerSim est un logiciel de simulation de protocoles pair à pair. Nous allons voir son principe de fonctionnement pour comprendre le travail effectué.

Tout d'abord, PeerSim ne va pas créer une véritable architecture pair à pair communiquant permettant la communication simultanée de plusieurs pairs. PeerSim va simuler seulement une interaction simultanée à l'aide d'un principe de cycle de fonctionnement. A chaque cycle, PeerSim va appeler la méthode nextCycle de chacun des pairs du réseau. Les pairs agissent les uns après les autres, mais on considère que toutes les actions effectuées durant un même cycle se font en réalité en simultané.

Pour fonctionner, le simulateur a besoin d'un fichier de config qui va regrouper les informations dont il a besoin. Il faut par exemple lui donner le nombre de pairs à créer pour le réseau, la classe que l'on souhaite utiliser pour les pairs, le protocole à utiliser, les relations de voisinage entre les pairs, etc… On a donc aussi une classe à créer pour implémenter les pairs à utiliser. Celle-ci doit hériter d'une classe mère définie dans PeerSim. Enfin, il est possible de créer des « observers » qui sont appelés à chaque fin de cycle. Cela permet d'effectuer un point sur le réseau à la fin de chaque cycle.

3. PROBLEMES RENCONTRES AVEC LE JEU DE DONNEES

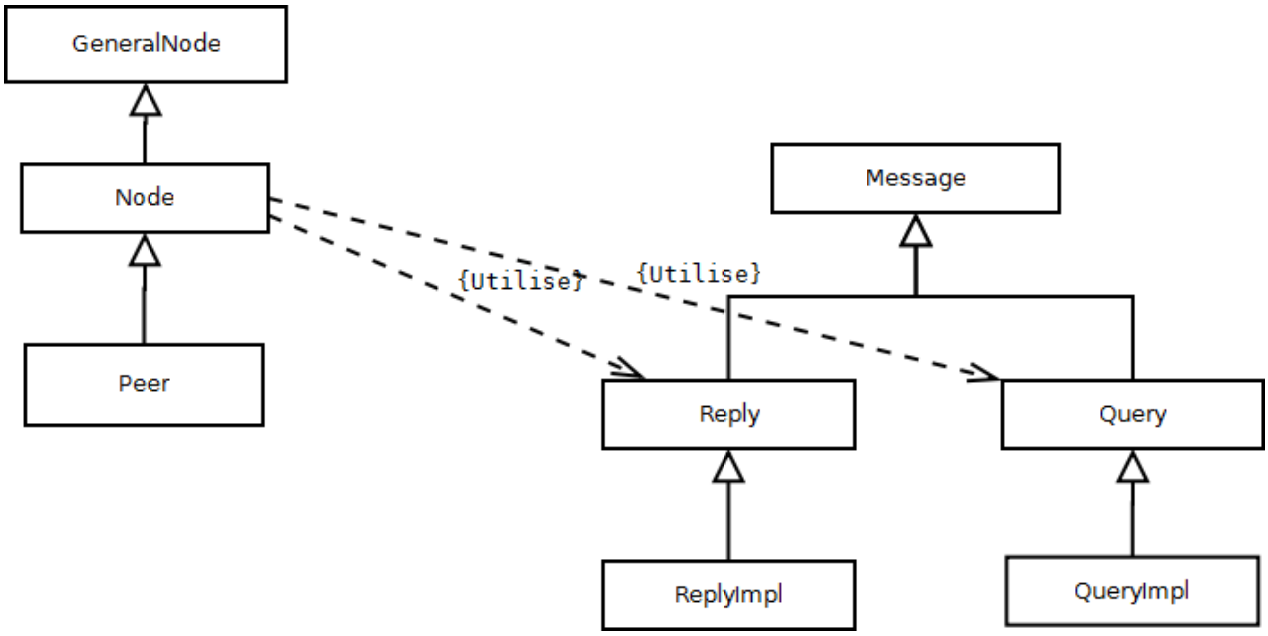
Le jeu de données récupéré pose quelques problèmes pour la simulation. En effet, le jeu de données contient énormément de données ce qui rend les simulations compliquées. Le jeu de données contient plus de 300 000 utilisateurs qui représentent donc chacun un pair dans la simulation. Il faudra à chaque cycle questionner chaque pair pour savoir s'il a une action à effectuer. Nous avons aussi plus d'un million de notes à stocker dans le simulateur pour effectuer la simulation. Pour des soucis de performance, on ne peut pas se permettre de ne pas stocker ces notes et de faire des appels en base de données pour les

récupérer. Toutes ces données représentent donc une place mémoire non négligeable pour le lancement de la simulation.

Le deuxième problème important dans la simulation vient du nombre de messages à échanger. Le but de la simulation est de tester pour chaque note d'un utilisateur, la note qu'il aurait obtenu en la demandant à l'aide de notre algorithme de recommandation. Plus la note obtenue est proche de celle recherché et plus notre algorithme est pertinent. Or avec plus d'un millions de note à tester cela représente un très grand nombre de message à échanger entre les pairs. L'algorithme procède de la manière suivante. Si l'on désire la note d'un objet, on va la demander à nos amis. S'ils ont la note recherchée, ils nous répondent sinon, ils transmettent eux même la requête à leurs amis jusqu'à une profondeur de n. Pour n=1, on ne demande que la note de nos amis directs. Avec cette profondeur, il y a assez peu de problèmes posés pourvu que l'on répartisse nos requêtes sur un nombre assez important de cycle. Mais en revanche, dès que l'on passe à une profondeur de 2 ou 3, le nombre de messages explose. Ces messages sont stockés pour être traités, mais le trop grand nombre de messages échangés entraine des soucis de performance. Il faut pour que la simulation fonctionne séparer les requêtes sur beaucoup plus de cycles.

4. IMPLEMENTATION

Dans cette partie nous allons voir les choix effectués pour l'implémentation de la simulation. Nous allons commencer par voir le diagramme de classe de la simulation pour comprendre l'architecture de la solution, puis nous verrons un diagramme de séquence afin de comprendre le déroulement de la simulation puis nous verrons les configurations possibles pour la simulation.

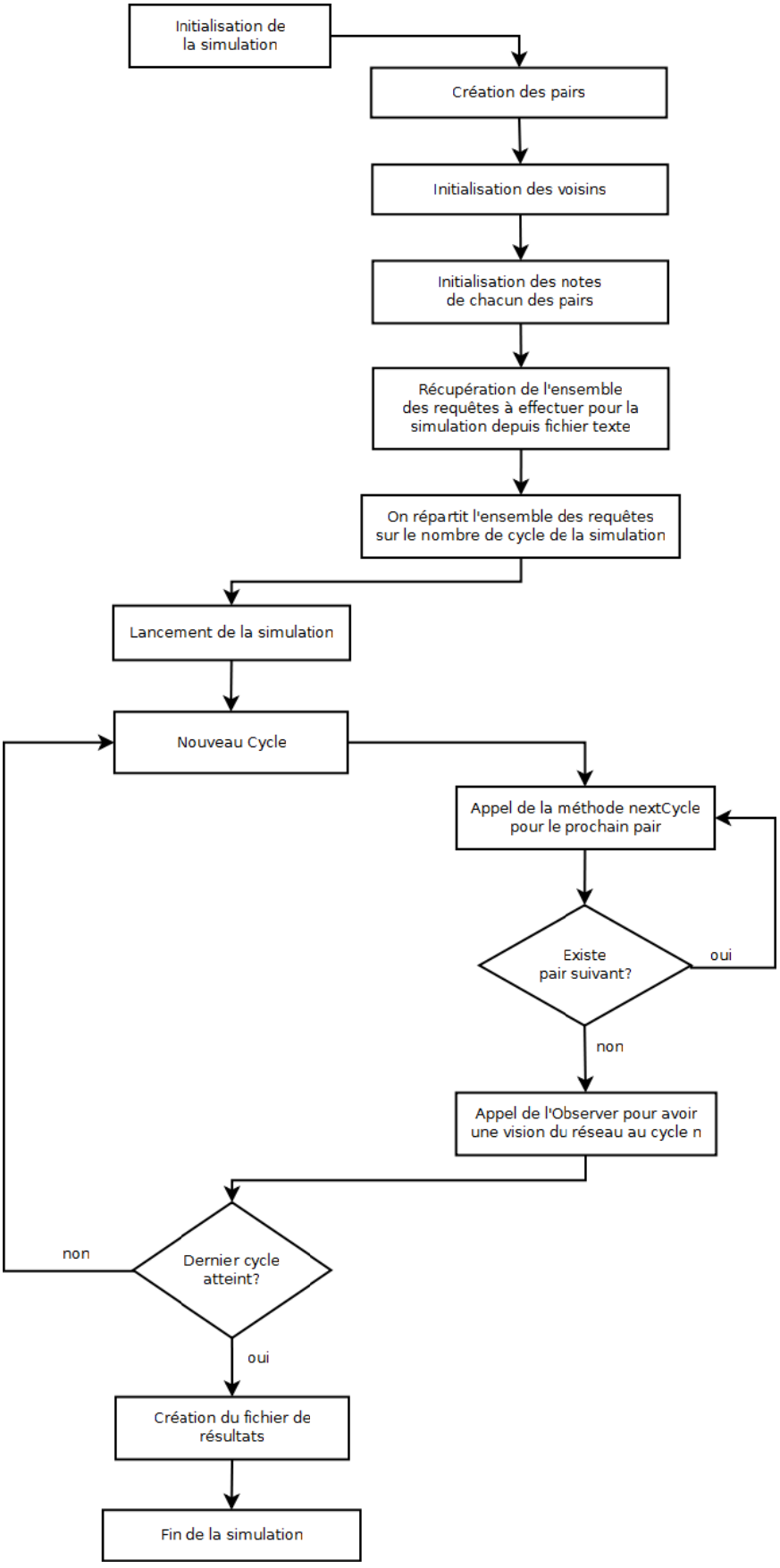


Ce diagramme présente une vue macro de l'architecture des pairs. Pour permettre une évolution facile du code, j'ai opté pour des classes abstraites tels que Node, Reply et Query. Cette façon de procéder me permet de changer l'implémentation de l'un ou l'autre des composants sans avoir à modifier entièrement le code. Ainsi la méthode appelée à chaque cycle n'a pas besoin de connaitre la réelle implémentation de ces classes. Pour ce faire, j'ai implémenté une factory pour la création des messages Reply et Query. C'est cette factory qui elle seule connait l'implémentation de ces deux classes abstraites. J'ai défini dans le fichier de config de la simulation deux paramètres qui contiennent le nom de la classe à implémenter.

Toujours dans l'optique d'évolution du code, j'ai implémenté de la même manière une factory qui va se charger de créer la classe de calcul de l'algorithme. La classe à utiliser pour la simulation est elle aussi à définir dans le fichier de config. Le design pattern factory est très important dans ce cas là puisque c'est l'algorithme de calcul qui sera le plus à même d'être modifié afin de permettre de tester successivement plusieurs algorithmes afin de tester les performances de chacun.

DEROULEMENT DE LA SIMULATION

Maintenant que l'on a vu en détail l'architecture des classes, nous allons passer au déroulement de la simulation. Pour ce faire, voici un schéma modélisant le déroulement de la simulation.



Sur le diagramme précédent, on peut voir toutes les étapes de la simulation. La première partie représente l'initialisation dont une grande partie est géré par PeerSim directement depuis le fichier de configuration. Ainsi, on va indiquer dans ce fichier, le nombre de Pairs à créer puis le fichier dans lequel sont stockés tous les liens entre pairs.

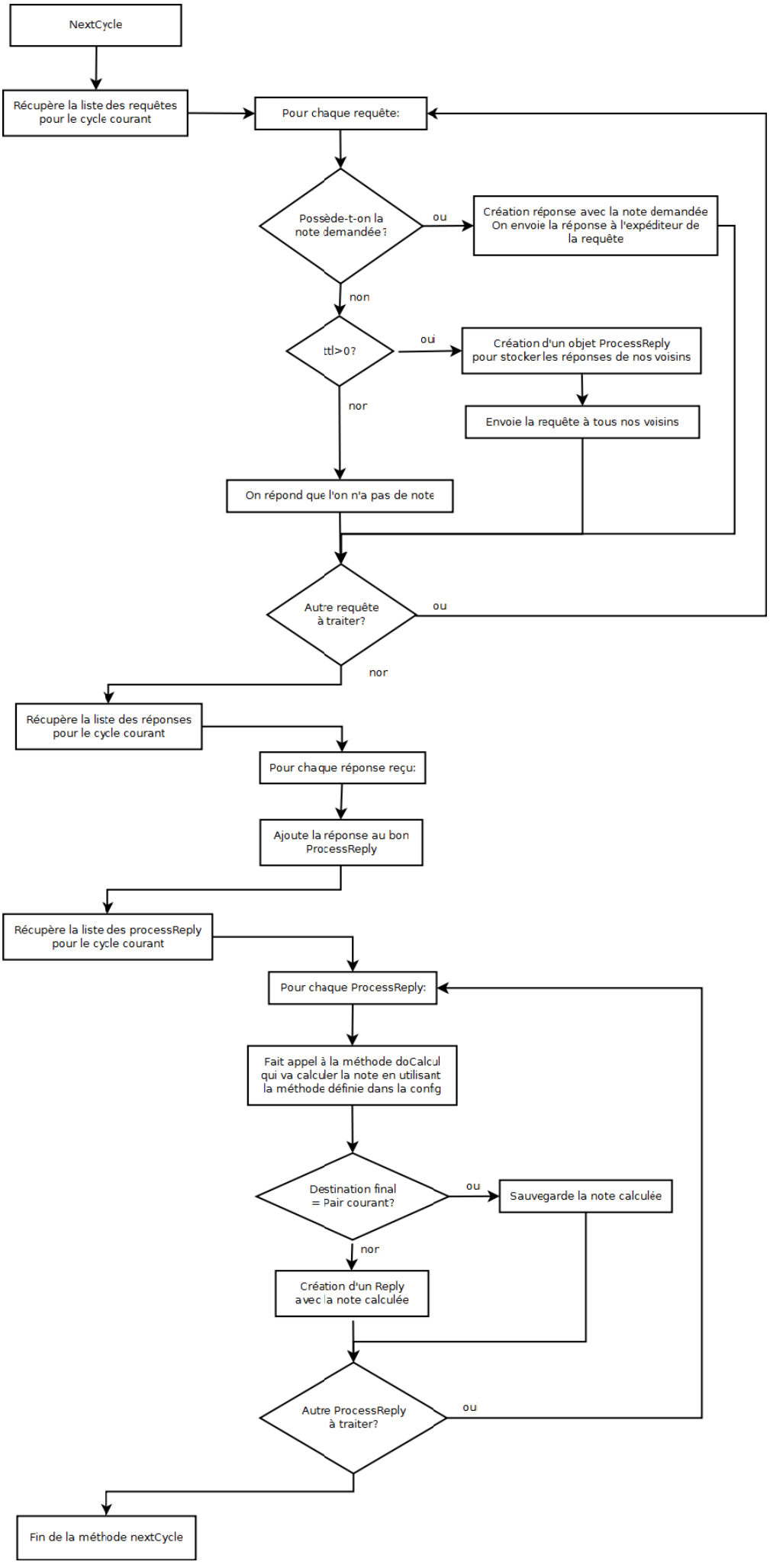
Puis lors de l'initialisation, on va récupérer dans la base de données l'ensemble des notes des utilisateurs et dans un fichier texte l'ensemble des requêtes que l'on désire voir durant la simulation.

Le schéma suivant va détailler la méthode clé de la simulation qui est la méthode nextCycle. Cette méthode est appelée à chaque cycle pour l'ensemble des pairs du réseau. C'est ici que l'on va détailler les actions de chacun des pairs pour chaque cycle. Comme nous pouvons le voir sur ce schéma, notre méthode va s'articuler autour de trois grands axes.

Le premier va concerner les requêtes à traiter pour le cycle courant. On définit au début de la simulation l'ensemble des notes que vont demander chacun des pairs en définissant aussi le cycle auquel le pair devra effectuer sa requête. Nous aurons donc pour chaque cycle un certain nombre de requêtes à traiter.

Si l'on est le créateur initial de la requête, on transmettra directement cette requête à tous nos voisins, sinon, on doit d'abord vérifier si l'on possède la note recherchée. Si c'est le cas, on peut répondre directement à l'expéditeur. Dans le cas contraire, les requêtes seront transmises aux voisins avec une profondeur de n définie en début de simulation. Cette profondeur est symbolisée par le ttl sur le schéma. A chaque nouvelle transmission du message, ce ttl sera décrémenter pour atteindre à 0 la profondeur recherchée. Pendant que les requêtes sont transmises aux pairs voisins, le pair courant doit stocker toutes les réponses des voisins et attendre qu'ils aient tous eu le temps de répondre pour pouvoir transmettre la réponse au demandeur initial. Il s'agit sur le schéma de la partie ProcessReply. On défini un temps d'attente égal à deux fois le ttl pour permettre aux messages de faire l'aller-retour entre le pair demandeur et la réponse la plus éloignée (profondeur de n).

C'est avec cet ensemble de notes récupérées dans les réponses que l'on va calculer la note qui doit être calculée par le pair courant. Dans la simulation, pour gagner en performance, on ne va pas tester si l'on a reçu l'ensemble des messages pour envoyer la réponse. On attend le nombre de cycle défini même si l'on a reçu l'ensemble des réponses.


NextCycle
Récupère la liste des requêtes pour le cycle courant
Pour chaque requête:
Possède-t-on la note demandée?
ou
Création réponse avec la note demandée On envoie la réponse à l'expéditeur de la requête
non
ttl>0?
oui
Création d'un objet ProcessReply pour stocker les réponses de nos voisins
Envoie la requête à tous nos voisins
nor
On répond que l'on n'a pas de note
Autre requête à traiter?
ou
nor
Récupère la liste des réponses pour le cycle courant
Pour chaque réponse reçu:
Ajoute la réponse au bon ProcessReply
Récupère la liste des processReply pour le cycle courant
Pour chaque ProcessReply:
Fait appel à la méthode doCalcul qui va calculer la note en utilisant la méthode définie dans la config
Destination final = Pair courant?
ou
Sauvegarde la note calculée
nor
Création d'un Reply avec la note calculée
Autre ProcessReply à traiter?
ou
Fin de la méthode nextCycle

5. RESULTATS

Maintenant que nous avons vu l'architecture et le déroulement de la simulation, il nous reste à voir les résultats obtenus. Comme expliqué précédemment dans la partie problèmes posés par le jeu de données, avec une profondeur de 2, la simulation n'est pas encore opérationnelle.

Je vais donc vous présenter les résultats obtenus avec une profondeur de 1. Pour l'instant seul un algorithme très simple de recommandation a été implémenté. J'ai pour le moment utilisé une simple méthode de moyenne entre les notes des voisins. D'autres algorithmes plus complexes seront implémentés plus tard.

QUELQUES CHIFFRES

Avec une profondeur de 1 :

- 1 127 742 requêtes à l'initialisation pour cette simulation
- Répartitions des requêtes sur 500 cycles
- 70 056 154 requêtes traitées durant la simulation
- 68 928 412 réponses envoyées durant la simulation
- 9 min de temps de simulation (initialisation incluse)
- 1 112 377 notes récupérées (moins que les requêtes pour cause de doublons)
- 262 493 notes non nulles

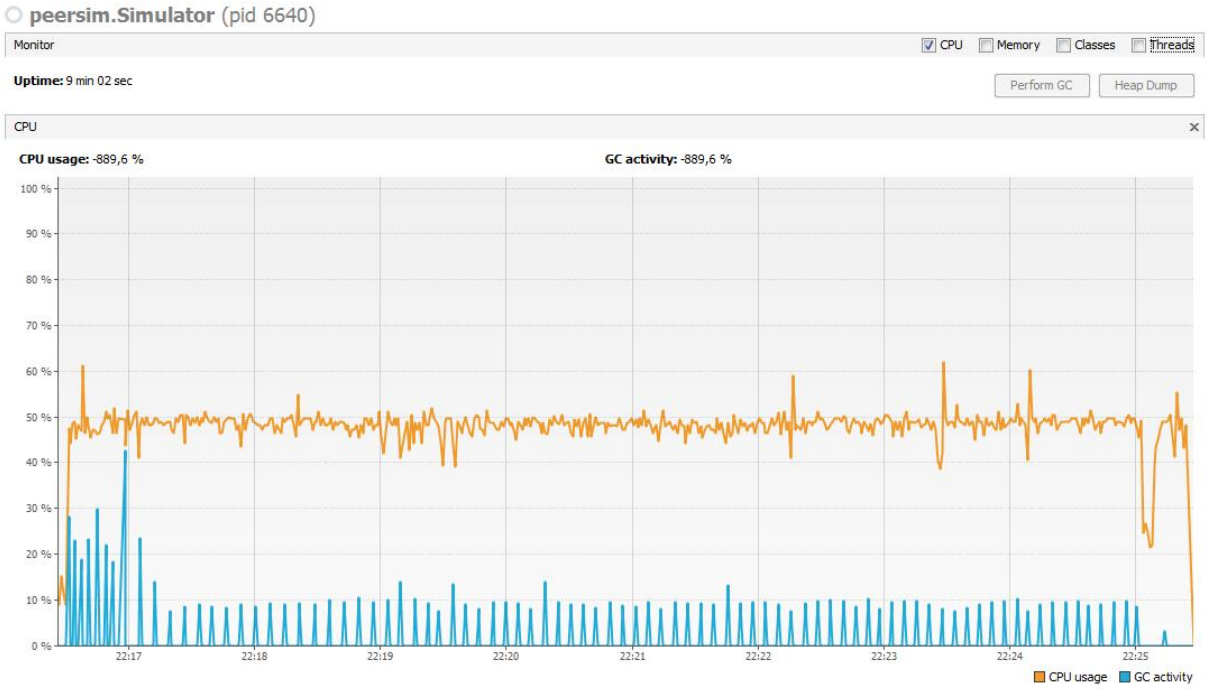
On n'a que 24% de notes récupérées non nulles dans cette simulation. Cela peut paraitre peu mais il faut voir que la simulation n'a été effectuée qu'à une profondeur de 1 donc en ne demandant les notes qu'aux voisins directs. Il faut aussi tenir compte du fait qu'une partie des utilisateurs n'ont pas de liens de confiance et ne peuvent donc pas obtenir de note. Ces 24% de notes non nulles sont en accords avec les résultats obtenus par Simon Meyffret dans ses travaux.

Pour une profondeur de 2, j'ai testé la simulation en répartissant les requêtes sur 5000 cycles, mais l'explosion du nombre de messages a tout de même empêché la simulation de se dérouler convenablement. En 2h20, la simulation n'en était qu'à 6%.

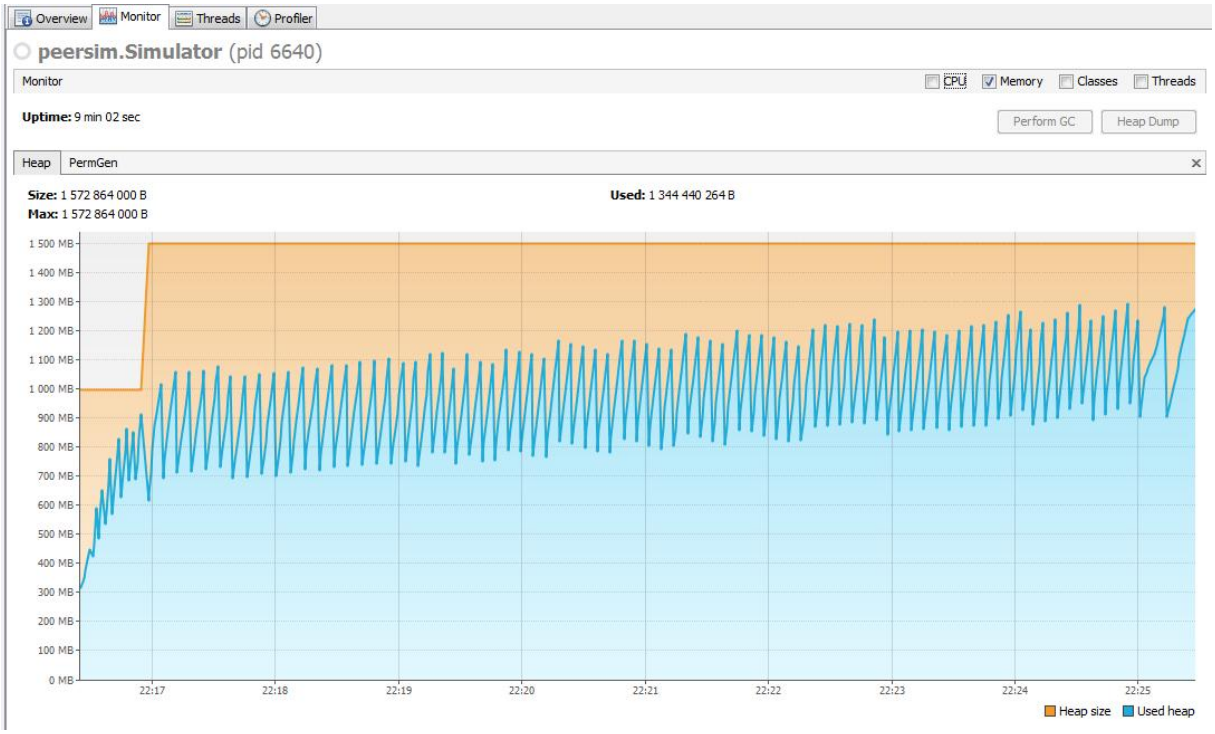
PERFORMANCES

Dans cette partie, il faut tenir compte du fait que les simulations ont été effectuées à l'aide d'un pc portable Intel Core 2 Duo GHz et 3Go de mémoire vive. Sachant que je n'ai pu allouer « que » 1,5 Go pour la simulation.

Voici les graphes récupérés à l'aide de JVisualVm à l'issue de la simulation pour une profondeur de 1 avec 500 cycles.



Sur ce diagramme, on observe que la simulation a pris 50% du CPU durant tout le temps d'exécution. En bleu, il s'agit du pourcentage CPU utilisé pour lancer le garbage collector et récupérer ainsi de l'espace mémoire pour l'exécution. On voit ici que ce temps n'est pas négligeable. Ceci est du au fait de la création très rapide de messages lors de la transmission de requêtes et les réponses de chacun des pairs aux requêtes. Ces messages ont pour autant une faible durée de vie et sont donc rapidement supprimés par le garbage collector.



Sur ce diagramme, on voit maintenant la taille mémoire occupée par la simulation. Après l'initialisation, on a déjà 700Mo de mémoire utilisée par la simulation. On observe à chaque pic l'action du garbage collector qui va supprimer les messages qui ne sont plus utilisés. On se rend tout de même compte que la mémoire croit durant la simulation. Une explication de cette croissance peut être que le stockage des notes calculées prend plus de place en mémoire que le stockage des requêtes initiales.

Ces deux diagrammes ont été effectués durant la même simulation pour une profondeur de 1 avec 500 cycles. J'ai voulu tester avec une profondeur de 2 en répartissant cette fois les requêtes sur 5000 cycles. Les deux schémas vus précédemment ont été tout à fait différents pour cette simulation. J'ai pu observer que la simulation passé quasiment tout son temps dans le garbage collector pour faire de la place mémoire. Ensuite, la taille mémoire occupée par la simulation a rapidement atteint 1,5Go et a stagné à ce niveau durant les deux heures d'exécution.

6. AMELIORATIONS ET PERSPECTIVES

Les améliorations que l'on peut apporter à la simulation sont encore très nombreuses. J'ai passé beaucoup de temps dans l'amélioration des performances de la simulation pour utiliser le moins de mémoire possible, mais cela n'a pas suffit pour la profondeur de recherche de 2. Il reste surement des améliorations à apporter de ce côté-là.

Une autre amélioration possible serait l'utilisation d'un pool de requêtes pour éviter d'avoir à allouer trop d'objets et du coup de réduire l'utilisation du garbage collector.

Pour l'instant, le jeu de données n'est pas encore épuré de ses imperfections. Il y a de très nombreux utilisateurs n'ayant pas de lien de confiance qui ne sont donc pas utiles aux simulations. Supprimer ces utilisateurs permettrait d'améliorer les performances de la simulation.

Au-delà des performances d'exécution, il reste à implémenter sur le simulateur des algorithmes de recommandation plus complexe que celui de la moyenne des voisins utilisé pour l'instant. Cette implémentation devrait s'intégrer très facilement au simulateur grâce au design factory utilisé pour la création de l'objet de calcul.

Dans les perspectives, il serait surement possible de lancer la simulation pour une profondeur de 2 avec une machine plus performante que celle sur laquelle j'ai pu travailler. Une fois la base épurée, et en choisissant un nombre de cycle plus important, on devrait pouvoir effectuer cette simulation.

CONCLUSION

Pour conclure sur ce rapport, je ne m'attendais pas au début de ce PFE à récupérer un jeu de données aussi important. Je pensais que cette partie serait assez simple et rapidement terminée, mais il s'est avéré que le site d'Epinions contient énormément de particularités et ne présente pas une structure HTML facile à analyser. J'ai donc rencontré sur cette partie beaucoup plus de problèmes qu'attendus, mais en même temps les résultats ont été aussi plus importants que prévu.

Pour la deuxième partie du PFE, j'avais commencé à travailler sur un prototype android, mais après une réunion durant laquelle nous avons revu les priorités il a été décidé de finalement commencer par la simulation à l'aide de PeerSim. Je n'ai malheureusement pas eu le temps de finir cette partie plus tôt pour me remettre à travailler sur la conception du prototype.

Pour la partie PeerSim, il y avait des enjeux différents de la première partie. En effet, il fallait faire un simulateur qui serait sujet à de nombreuses évolutions. J'ai donc passé plus de temps sur la conception et l'optimisation de mes solutions. Ce fût une expérience très instructive pour l'amélioration de mes méthodes de développement. Au niveau des résultats, le simulateur n'est pas encore totalement opérationnel car il manque encore l'implémentation de la méthode de calcul de la recommandation, mais la structure de la simulation est prête.

Ce PFE m'a permis de découvrir le milieu de la recommandation ainsi que ce qu'est la recherche. Il m'a aussi permis d'améliorer mes compétences techniques, surtout au niveau de l'optimisation du code et de la réutilisabilité.

BIBLIOGRAPHIE

Site d'Epinions :

- http://www.epinions.com/

Aspirateur de site :

- http://fr.wikipedia.org/wiki/Aspirateur_de_site_Web
- http://en.wikipedia.org/wiki/Web_crawler
- http://www.idf.net/info/table.html

Article de comparaison parseurs html

- http://www.benmccann.com/dev-blog/java-html-parsing-library-comparison/

TagSoup :

- http://home.ccil.org/~cowan/XML/tagsoup/
- http://www.hackdiary.com/2003/04/13/screenscraping-html-with-tagsoup-and-xpath/

HTMLCleaner :

- http://htmlcleaner.sourceforge.net/
- http://thinkandroid.wordpress.com/2010/01/05/using-xpath-and-html-cleaner-to-parse-html-xml/

XPath :

- http://fr.wikipedia.org/wiki/XPath
- http://www.w3.org/TR/xpath/
- http://jerome.developpez.com/xmlxsl/xpath/
- http://zedros.developpez.com/tutoriels/java/xml/xpath/

Expressions régulières :

- http://en.wikipedia.org/wiki/Regular_expression

Base de données :

- http://www.mysql.fr/
- http://www.postgresql.org/
- http://www.siteduzero.com/tutoriel-3-31600-mysql-et-postgresql-lequel-choisir.html

- http://www.wikivs.com/wiki/MySQL_vs_PostgreSQL

Httrack :

- http://www.httrack.com/
- http://www.httrack.com/html/shelldoc.html

Sip2Peer:

- http://code.google.com/p/sip2peer/
- http://sip2peer.googlecode.com/files/s2p-Tutorial-Bootstrap-FullPeer-SBC.pdf

AllJoyn:

- https://www.alljoyn.org/
- https://www.alljoyn.org/sites/default/files/80-BA001-1_C_AllJoyn-Android-NDK-Developer-Guide.pdf
- http://www.generation-nt.com/alljoyn-qualcomm-communications-mobiles-p2p-open-source-actualite-1156641.html

Frostwire

- http://www.frostwire.com/
- http://fr.wikipedia.org/wiki/FrostWire

PeerSim

- http://peersim.sourceforge.net/
- http://peersim.sourceforge.net/tutorial2/tutorial2.pdf
- http://www.metz.supelec.fr/metz/personnel/galtier/PagesPerso/Enseignement/3A/SOA/P2P/tpP2P.html