Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Projet Recherche & Développement
2015-2016

Développement d'un web crawler pour la protection des mangas sous copyright

Tuteurs académiques
Mathieu Delalandre

Étudiants
Aurélien PLANCHON (DI5)

3 février 2016

Liste des intervenants

Nom	Mail	Qualité
Aurélien PLANCHON	aurelien.planchon@etu.univ-tours.fr	Étudiant DI5
Mathieu Delalandre	mathieu.delalandre@univ-tours.fr	Tuteur académique, Département informatique

Avertissement

Ce document a été rédigé par Aurélien PLANCHON susnommé les auteurs.

L'école polytechnique de l'université François Rabelais de Tours est représentée par Mathieu Delalandre susnommé les tuteurs académiques.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

Les auteurs reconnaissent assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respects des lois ou des droits d'auteur.

Les auteurs attestent que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

Les auteurs attestent ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

Les auteurs attestent que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

Les auteurs reconnaissent qu'ils ne peuvent diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable des tuteurs académiques.

Les auteurs autorisent l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Pour citer ce document :
Aurélien PLANCHON, *Développement d'un web crawler pour la protection des mangas sous copyright*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2015-2016.

```
@mastersthesis{
    author={PLANCHON, Aurélien},
    title={Développement d'un web crawler pour la protection des mangas sous copyright},
    type={Projet Recherche & Développement},
    school={Ecole Polytechnique de l'Université François Rabelais de Tours},
    address={Tours, France},
    year={2015-2016}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document `polytech.cls` V1.0 (2015/11/18)

Table des matières

Table des figures

7 Campagne de tests

Liste des tableaux

Introduction

Un *web crawler* ou *web collecteur* en français, est un logiciel qui explore des sites web à la recherche de contenu ou simplement pour effectuer une indexation [WWW4].

1 Contexte

L'objectif de ce projet est de réaliser un webcrawler qui sera capable de parcourir et télécharger automatiquement des banques d'images. Ce projet se focalise sur 3 sites différents présentant des configurations différentes.

Table 1 – *Comparaison volumétrique des banques d'images*

Site web	Adresse	Nombre de Téléchargements	Volume	Commentaires
DCM THE DIGITAL COMIC MUSEUM	Lien	17.100	7 To	17.100+ œuvres à télécharger sous forme d'archive *.cbz*[WWW7]
ComicBook+ ©2015 ComicBookPlus.com	Lien	26.960	400 Go	26.960 œuvres à télécharger sous forme d'archive
mangapanda	Lien	10.120k	3 To	4042 œuvres dont le chiffre ne fait qu'augmenter. Les images sont à télécharger une par une en parcourant le site.

La phase de développement sera décomposée en 4 parties.

— La première partie sera la mise en place des fonctions nécessaires au web crawler pour parcourir un site web.
— La seconde partie sera la création d'un downloader.
— La troisième partie sera la mise en place des optimisations dans l'utilisation des communications réseau.
— Et enfin la dernière partie sera la mise en place d'un ordonnanceur efficace pour gérer le processus global du web crawler.

Première partie

Recherche

1 État de l'art

1 Webcrawler

Un webcrawler est un logiciel parcourant les sites web à la recherche d'informations. L'exemple le plus parlant de webcrawler sont les robots d'indexation qui permettent le référencement des sites web sur les moteurs de recherches.

1.1 Fonctionnement

Un *web crawler* fonctionne de manière simple. Il part d'une page web donnée que nous appellerons *page source*, en explore le contenu, et parcourt tous les liens de la page en question de manière récursive. Évidemment afin de ne pas retourner sur des pages déjà explorées il garde en mémoire les url déjà explorées. Voir Figure 1.

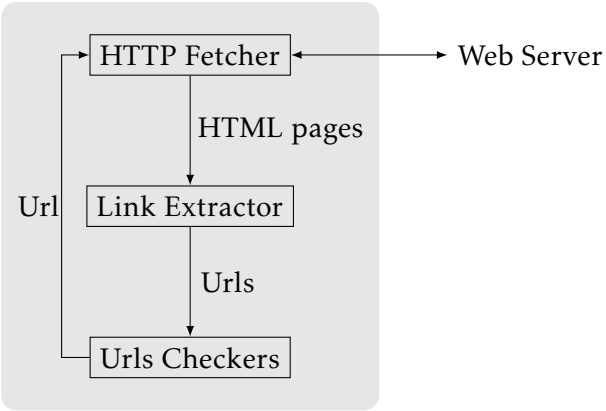


Figure 1 – *Fonctionnement basique d'un web crawler*

Mais tout n'est pas si simple. Pour commencer, l'url [1] passée est très souvent un nom de domaine. La requête HTTP initiatrice du module *HTTP Fetcher* sera donc redirigée vers un DNS [2]. On peut donc rajouter cette partie au schéma. Voir Figure 2.

1. Uniform Ressource Locator : désigne une chaine de caractère utilisée pour adresser les ressources web.[WWW6]
2. Domain Name System : système permettant de transformer un nom de domaine en une adresse IP

De même la partie *Urls Checkers* est beaucoup plus complexe, elle doit :

- Supprimer les liens répertoriés en double sur une page
- Supprimer les liens déjà visités par le web crawler
- Transformer les liens relatifs en liens complets
- Ajouter chaque URL à une liste triée par priorité sur laquelle le module *HTTP Fetcher* viendra récupérer la nouvelle URL

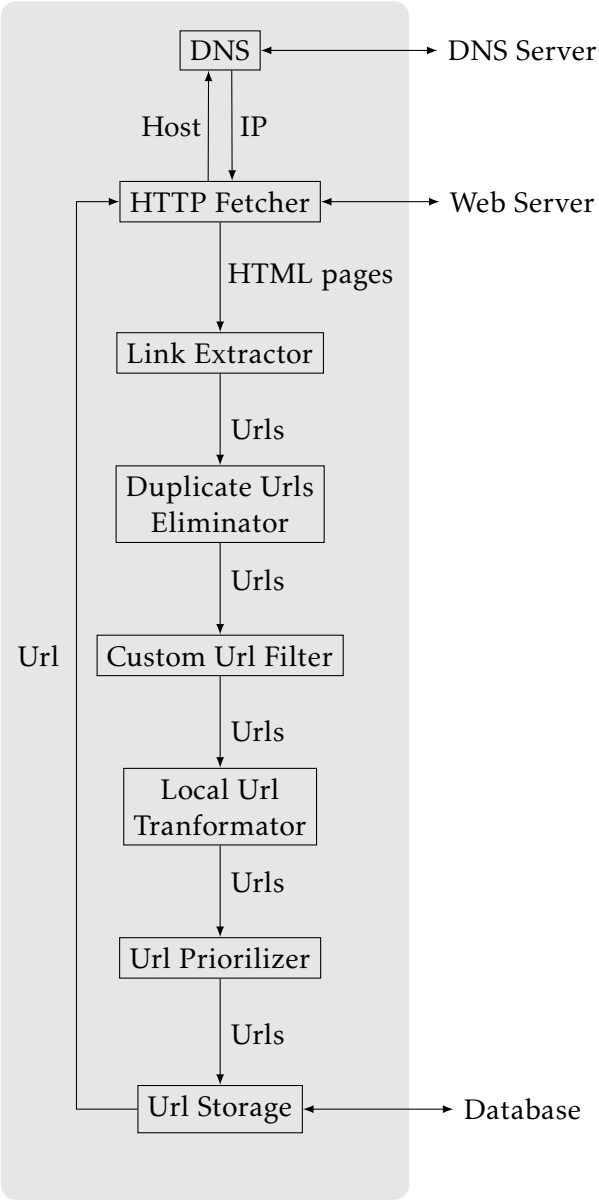


Figure 2 – *Fonctionnement d'un web crawler*

Les *web crawlers* sont donc des *bots* [3]. Il existe de nombreuses problématiques liées à l'utilisation de *bots* sur internet. Une des problématiques vient de la vitesse à laquelle le système envoi une requête à un site web. Il est nécessaire de vérifier que l'utilisation d'un tel dispositif n'impactera pas le site web. Un nombre de requêtes trop important pour un serveur revient à effectuer une attaque DoS [4] écroulant le site web ou bien submergeant le réseau avec les requêtes trop nombreuses. Un autre aspect, par exemple

3. Robots
4. Denial of Service

est l'utilisation de captcha (généralement utilisé pour la création de compte) afin d'éviter qu'un *bot* puisse s'y inscrire automatiquement. Il existe d'autres problématiques liées à l'aspect sécuritaire tel que la triche dans les jeux en ligne, etc ...

L'ensemble des informations liées à l'état de l'art sur le fonctionnement des web crawler se retrouve dans ce document : [4]

1.2 Utilisations des protocoles de transmission

Pour fonctionner le webcrawler a besoin de télécharger du contenu. Pour ce faire il se base sur le protocole HTTP et les connexions TCP. Depuis 1999 la norme HTTP 1.1 est de mise. La norme initialement décrite par le RFC 2616 a été retravaillée en juin 2014 afin de clarifier sa spécification et son fonctionnement. HTTP 1.1 est un protocole stable. Ce sera le protocole utilisé étant donné sa large implémentation. Il a comme avantage de pouvoir garder une connexion TCP ouverte avec le serveur afin d'envoyer plusieurs requêtes sans avoir à recréer une connexion TCP à chaque fois. Ces connexions persistantes sont réalisables en utilisant l'entête **Connection : Keep-Alive** [WWW2]. Depuis mai 2015 la norme HTTP/2 remplace la version 1.1, mais cette norme très récente n'étant pas supportée par l'intégralité des navigateurs ni des serveurs web. Nous restons sur la version 1.1 pour le projet.

2 Scheduling

L'ordonnancement de téléchargement de page web est un aspect très important dans l'optimisation des webcrawlers. L'optimisation s'appuie essentiellement sur l'augmentation de la vitesse de téléchargement ou sur l'ordonnancement des pages à télécharger (prioriser les pages ayant du contenu de valeur).

2.1 Ordonnancement à deux niveaux

[3] explique très clairement les méthodes d'optimisation des webcrawlers. Il est mis en évidence qu'une implémentation efficace d'un webcrawler doit être faite avec un ordonnancement à deux niveaux. Ainsi un *Scheduler* doit prendre en charge l'ordonnancement des sites web à parcourir, et pour chaque site web les pages à parcourir sont elles aussi ordonnancées. Ce qui peut être représenter par la figure : 3. Les *urls* à parcourir sont ordonnancées par priorité et associées à leurs site web. Et les sites web sont eux aussi ordonnancés selon leurs priorités. Les priorités sont définies en fonction des stratégies du *Webcrawler*.

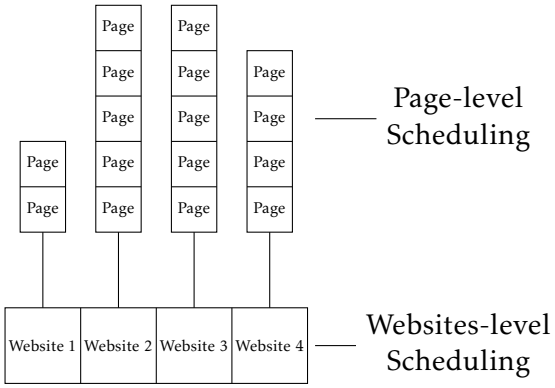


Figure 3 – *Ordonnancement à deux niveaux*

2.1.1 Stratégies de crawling

La littérature sur les stratégies de webcrawling se base essentiellement sur la priorisation du contenu à récupérer. Par exemple avec des stratégies reposant sur le *PageRank* [5].

5. *"Le **PageRank** est un algorithme qui mesure quantitativement la popularité d'une page web."*[WWW3]

Il existe d'autres stratégies [3] comme :

- **Parcours en largeur :** *(Breadth-first scheduling)* Les urls sont parcourues par ordre de découverte, et les serveurs sont ordonnancés selon une file FIFO [6].
- **Priorisation de performance :** *Performance based scheduling* Cet algorithme a comme objectif de parcourir un maximum de pages dans un temps donné. Il priorise les connexions rapides pour récupérer un maximum de contenu. Dans ce contexte la **Server Queue** bénéficie d'un ordonnancement spécifique mais les *Url Queues* fonctionnent toujours en FIFO.
- **Priorisation de qualité :** *Quality based scheduling* Cet algorithme se base sur l'évaluation de la qualité du contenu à récupérer (évaluation faite en amont du téléchargement). Les *Url Queues* sont ordonnancées selon la qualité de l'Url et la *Server Queue* est ordonnancée selon la qualité de la meilleure Url du serveur.
- **Priorisation Hybrid :** *Crawl-ability based scheduling* Cet algorithme combine *Qualité* et *Performance*. On définit un critère *crawl-ability* pour les serveurs se basant sur le temps de réponse et la qualité des pages web. Les serveurs sont classés selon ce critère et les urls selon leur qualité dans chaque queue.

En ce qui concerne le présent, l'algorithme qui sera implémenté et modifié pour s'adapter à notre objectif est un algorithme se basant essentiellement sur la priorisation de performances. L'objectif n'est pas de récupérer que certaines informations mais la totalité du contenu image. L'objectif n'est pas non plus de récupérer un maximum de contenu en un minimum de temps mais simplement d'optimiser la bande passante via un ordonnancement. Le processus est tout particulièrement illustré dans ce papier : [WWW1].

Le principe est de noter les sites web avec la formule ci dessous :

$$R(s,i) = \frac{P(s,i)}{T(s,i)} \tag{1}$$

$R(s,i)$ étant la note pour un serveur web s lors de la connexion i. $P(s,i)$ représente le nombre de pages à télécharger depuis le serveur s lors de la connexion i (dans notre cas, ce ne sont pas des pages HTML mais archives ou images). Et $T(s,i)$ le temps nécessaire pour effectuer le téléchargement.

On peut remarquer tout de suite un problème : $P(s,i)$ et $R(s,i)$ ne peuvent être déterminés qu'après le téléchargement. Pour effectuer un ordonnancement efficace nous avons besoin de déterminer ces valeurs à priori. Pour ce faire nous utilisons ces formules :

$$P(s,i) = min\{RequestsPerConnection(s,i), TaskURLs(s,i)\} \tag{2}$$

$$T(s,i) = 2 * ConnectionTime(s,i) + \frac{P(s,i) * ResponseTime(s,i)}{p} \tag{3}$$

Le nombre de pages est fixé comme étant le minimum entre le nombre d'URL à télécharger lors de la connexion (estimée lors du téléchargement précédent) et le nombre maximum d'URL stockables. Le temps de téléchargement est définis comme étant le temps d'ouverture et fermeture de la connexion $2 * ConnectionTime(s,i)$ et le total de temps pour récupérer le nombre de pages. Le facteur p est un paramètre permettant prendre en compte les serveur Web supportant HTTP 1.1 pipelining [7]. Ne pouvant toujours pas déterminer correctement $ConnectionTime(s,i)$ et $ResponseTime(s,i)$ nous réalisons une prédiction temporelle.

$$ConnectionTime(s,i) = ConnectionTime(s,i-1) * \alpha + MeasureConnectionTime(s,i-1) * (1-\alpha) \tag{4}$$

$$ReponseTime(s,i) = ResponseTime(s,i-1) * \alpha + MeasureResponseTime(s,i-1) * (1-\alpha) \tag{5}$$

Nous pouvons donc déterminer à tout instant une note pour la connexion à venir en nous basant sur l'état de la dernière connexion.

6. First In First Out

7. Le pipelining HTTP est une technique consistant à combiner plusieurs requêtes HTTP dans une seule connexion TCP sans attendre les réponses correspondant à chaque requête. Source

Le paramètre α est un paramètre qui permet de choisir entre : donner plus d'importance à l'état de la dernière connexion, ou donner plus d'importance à l'intégralité de l'historique de l'état de la connexion avec ce serveur web. Lorsque $\alpha \in [0,1]$ augmente l'historique a plus de poids et lorsqu'il diminue c'est l'état le plus récent qui est pris en compte.

2 Optimisation réseau

La taille de la fenêtre offerte par le receveur peut être contrôlée par le processus de réception. Cela peut interargir sur les performances TCP[7]

Dans le cas d'une communication TCP/IP entre 2 clients, chaque client instancie 2 *buffers* de communication : un *buffer* d'envoi et un *buffer* de réception. Pour l'échange de fichier c'est la taille du *buffer* d'envoi du client source et le *buffer* réception du client destination qui comptent. La taille par défaut de 4096 bytes pour ces deux *buffer* n'est pas optimale dans ce cas d'utilisation. Nous allons donc interagir sur la taille de ces *buffers* afin d'augmenter la vitesse de téléchargement en fonction du débit de la communication. *Remarque : il existe aussi pour le client et le serveur un buffer applicatif. De préférence, ce buffer doit être de la même taille que le buffer d'envoi et/ou de réception, selon l'utilisation faite par l'application, afin d'éviter les goulots d'étranglement.*

Nous ne pouvons agir que sur le *buffer* du socket de destination, celui du client source étant le serveur web depuis lequel nous téléchargerons l'image. "*Des buffers de taille supérieure permettent d'utiliser les capacités de connexions à haut débit. Cela réduit le nombre d'écritures physiques sur le réseau, et amortis le coût d'écriture des 40 bytes liés aux en-têtes TCP et IP.*"[5]. Pour ce faire, il faut changer la taille de réception du buffer du client avant l'établissement de la communication entre le client et le serveur. Ainsi le serveur pourra adapter sa taille de buffer d'envoi en fonction de la taille du buffer de réception du client. Si une taille supérieure à 64Kb est précisée durant la séquence de connexion le mode *Windows sacling* sera alors activé.

Maintenant que nous avons déterminé comment activer l'option *window scaling* il est nécessaire d'expliquer son fonctionnement. Lors d'une communication TCP/IP les échanges sont validés par des acquittements. Lorsqu'une station envoi 64Kb de données, elle attend l'acquittement de la station destination qui confirme la réception des 64Kb. Cette taille c'est la **window**. Prenons un exemple :

L'activation du *windows scaling* permet d'augmenter l'entête qui définit la taille de la *window*. Ne pouvant pas augmenter sa taille de manière triviale (la taille d'une trame ne peut pas être aisément changée) une astuce permet d'émuler une augmentation de la taille : le décalage de bit. En effet en paramétrant lors de l'initialisation de la communication un décalage de bit, nous pouvons préciser une taille de *window* supérieur à 64Kb. Ce qui nous permet d'éviter des attentes de validation de synchronisation entre le client et le serveur et donc de télécharger plus vite. Par contre, en cas de pertes sur le réseau ce paramètre fera chuter au contraire la vitesse de téléchargement. C'est pourquoi il est nécessaire de l'appliquer dans un environnement performant avec une qualité de ligne entre le client et le serveur haute.

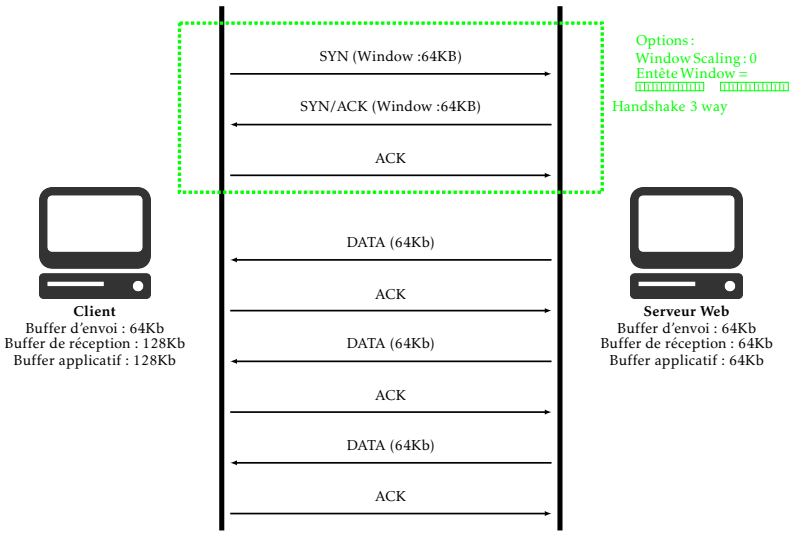


Figure 1 – *Téléchargement sans Window Scaling*

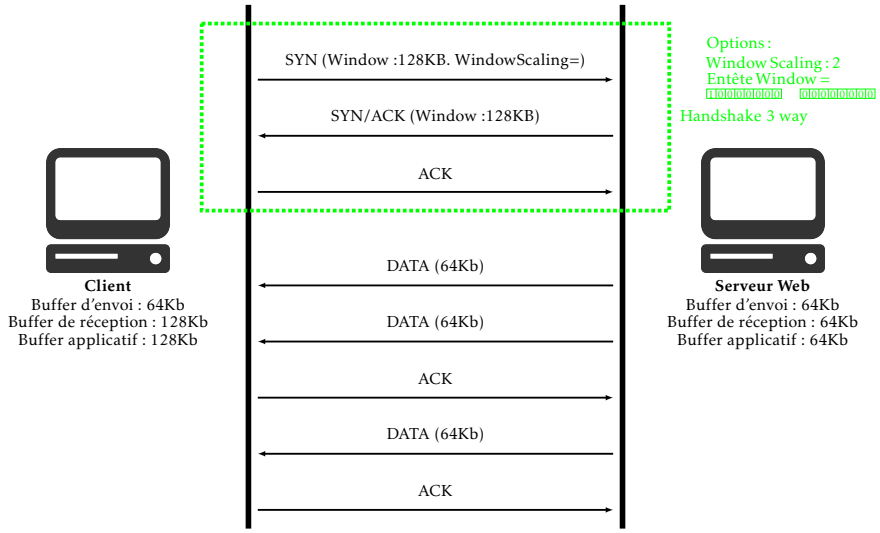


Figure 2 – *Téléchargement avec Window Scaling*

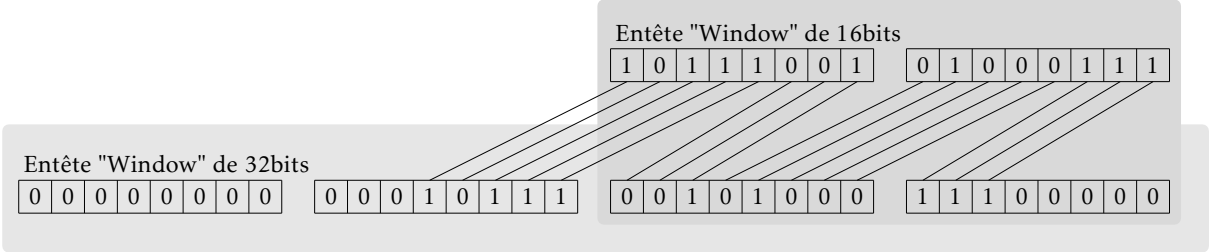


Figure 3 – *Window Scaling : shifting de 4*

3 Synchronisation

1 Pérennité des liens

Un problème qui pourrait se soulever lors de la collection des liens est l'obsolescence de ceux ci.

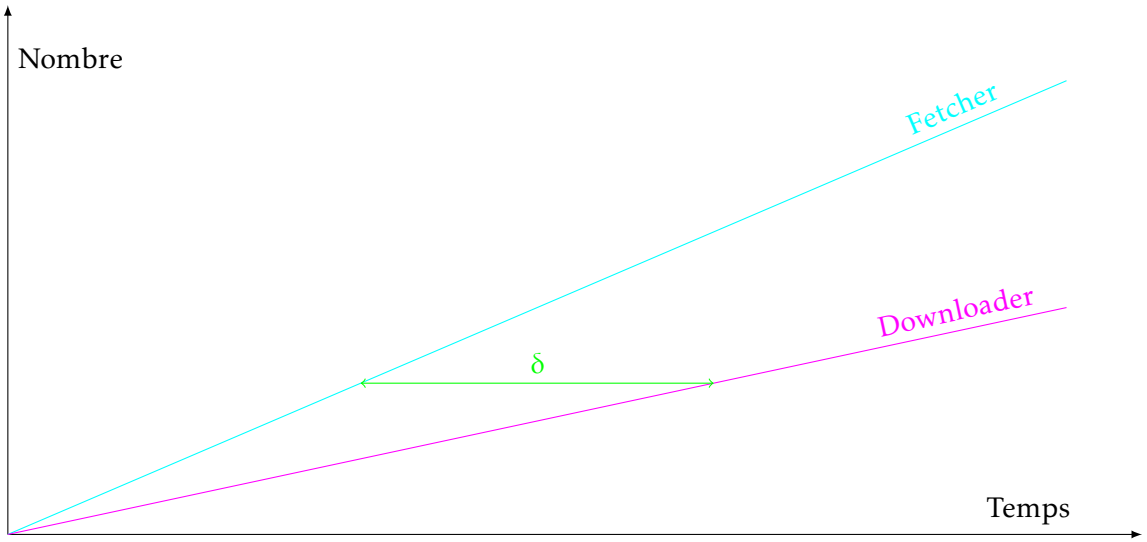


Figure 1 – *Comparaison de la vitesse de traitement Fetcher vs Downloader*

Comme nous pouvons observer sur le schéma, le module de collection des liens est bien plus rapide que celui de téléchargement (déduction simple due à la différence de taille entre le téléchargement du contenu HTML d'une page et d'un fichier image ou d'une archive). Il est donc nécessaire de mettre en place un système qui est capable d'assurer que les liens source qui sont téléchargés aient été collectés récemment. Pour ce faire, l'idée est d'alterner la collecte et le téléchargement afin de diminuer ce temps. (Temps noté δ sur la figure 1)

Une solution serait d'alterner l'exécution du *Fetcher* et du *Downloader* comme décrit figure 2

2 Producteur Consommateur

Nous nous retrouvons donc avec un problème producteurs/ consommateurs qui accèdent à des zones parfois protégées. Nous devons donc mettre en place une exclusion mutuelle pour protéger les sections

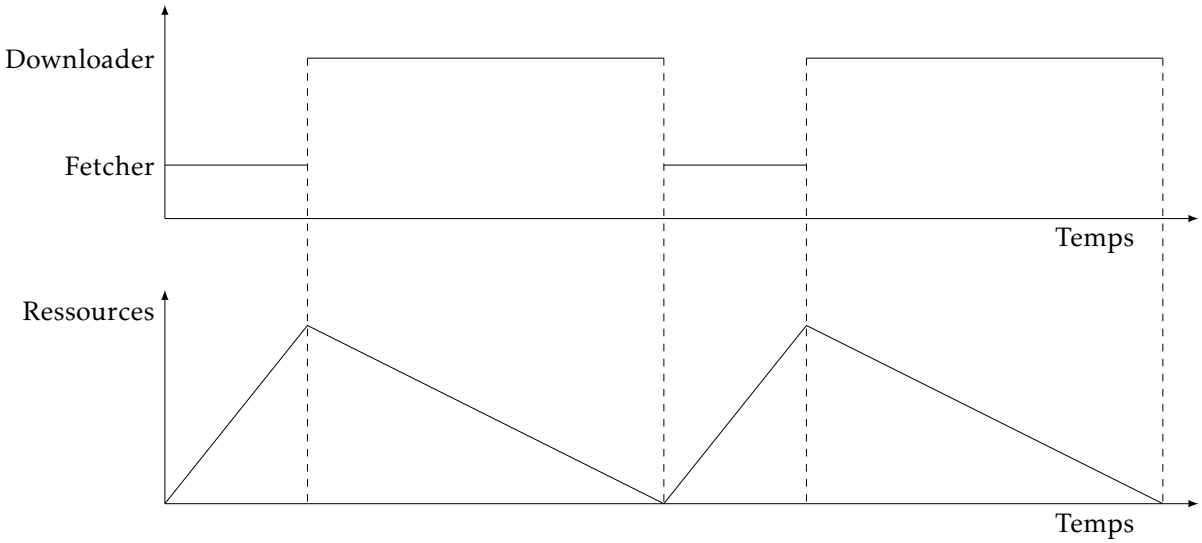


Figure 2 – *Alternance Fetcher-Downloader*

critiques du système, solutionner le problème producteurs consommateurs d'une manière à assurer l'alternance entre producteur et consommateur (comme décrit dans la section Pérennité des liens). Nous avons donc les *Fetchers* qui jouent le rôle de producteur en venant ajouter des liens à la BDD. Et des *Downloaders* qui sont les consommateurs et qui viennent télécharger les liens de la base de données.

3 Les moniteurs

Un moniteur est une solution aux problèmes d'exclusion mutuelle de programmation [6]. L'utilisation de sémaphores et de mutex pour la protection de sections critiques n'est pas toujours adaptée et peut très souvent amener d'importants ralentissements. Les moniteurs permettent de protéger l'accès à une section critique mais aussi de résoudre le problème producteurs consommateurs de manière programmatoire. La résolution de ce problème par un moniteur amène à alterner de manière plus ou moins régulière (critère arbitraire) l'appel au producteur et au consommateur. Ainsi un consommateur ne pourra pas être appelé s'il n'existe aucune ressource à consommer et un producteur s'arrêtera lorsque le nombre de ressources sera suffisant pour qu'un consommateur puisse les prendre en charge (critère arbitraire lui aussi).

Fonctionnement

Les moniteurs ont plusieurs types d'implémentation. Nous étudions tout particulièrement l'implémentation MESA [1] car elle correspond mieux à notre problème et qu'elle est supportée par le langage de programmation choisi pour le projet.

Lorsqu'une section est protégée par un moniteur comme décrit sur la Figure 3 et qu'un processus souhaite y accéder il rentre dans la queue des processus en attente d'accès au moniteur. Si celle-ci est vide et qu'aucun processus n'occupe la zone d'exécution il passe alors en zone d'exécution et accède à la section critique. Sinon il attend son tour dans cette première file d'attente. Prenons maintenant le cas particulier producteur/consommateur. Un moniteur fonctionne avec 2 variables **full** et **empty** avec une queue associée à chaque variable comme représenté Figure 3. Si un processus consommateur arrive dans la section critique et qu'il n'y a pas de ressource à consommer, il ira dans la file **emptyQ**. La file **emptyQ** ne sera libérée (i.e. les processus présents ne retourneront dans la queue d'accès au moniteur) que lorsqu'un producteur aura ajouté des ressources jusqu'à ce qu'il ne soit plus possible d'en rajouter. Les producteurs iront alors dans **fullQ** et tous les processus présents dans la file **emptyQ** retourneront dans la queue des processus en attente d'accès au moniteur.

1. Initialement l'implémentation était réalisée avec le langage MESA, elle en a gardé le nom.

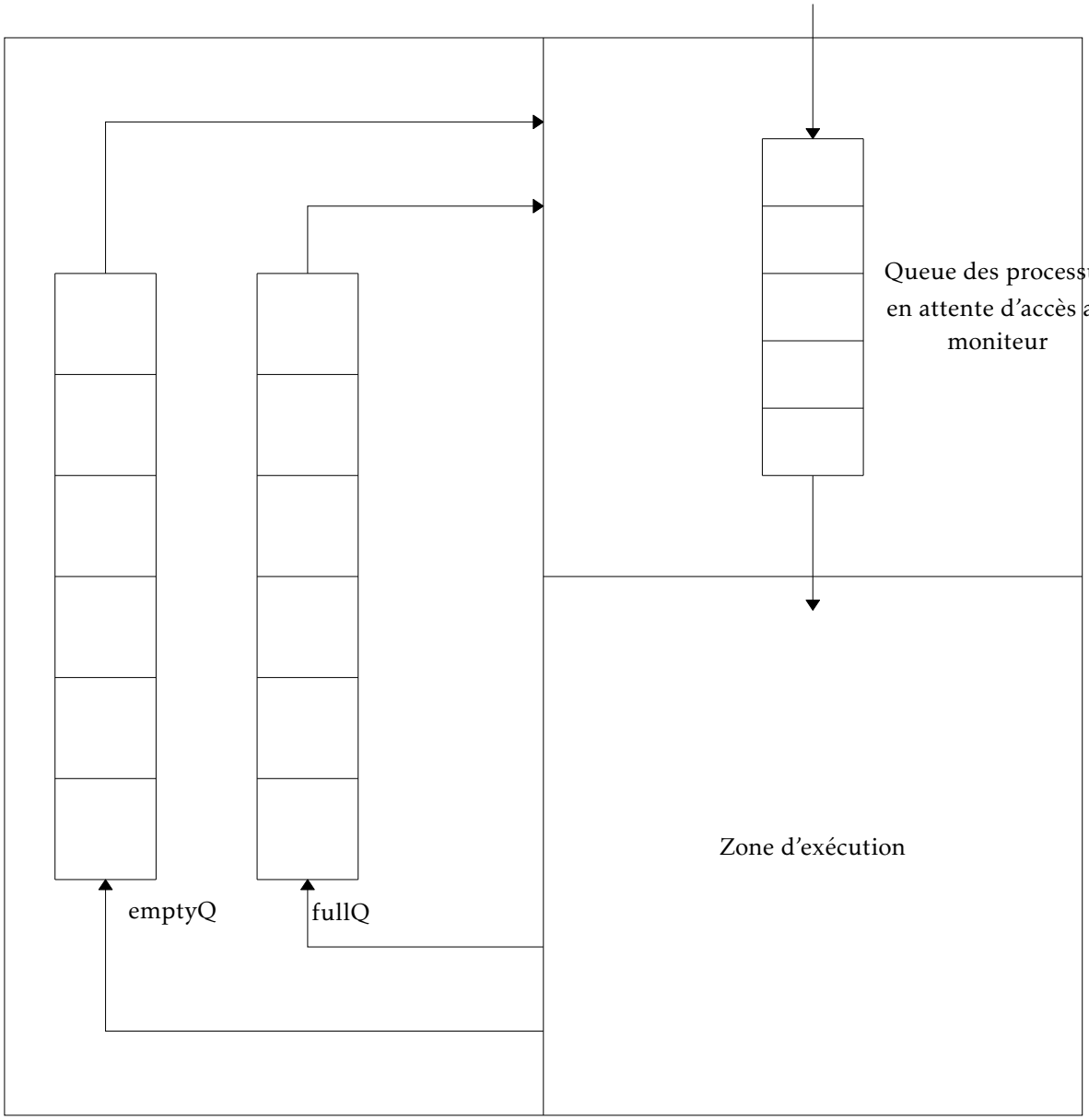


Figure 3 – *Représentation d'un moniteur de type Mesa*

On remarque donc que le moniteur est un moyen très efficace de résoudre le problème producteur consommateur et qu'il permet d'assurer que ni le producteur ni le consommateur ne soit appelé plus de fois que nécessaire.

Voici l'implémentation en Java d'un moniteur d'après [6] :

```
1 /* program producerconsumer */
2 monitor boundedbuffer;
3 char buffer [N]; // Space for N items
4 int nextin = 0;  // Buffer pointer
5 int nextout = 0; // Buffer pointer
6 int count = 0;   // Number of items in buffer
7 cond notfull, notempty; // Condition variables for synchronization
8
9 void append (char x){
```

```
10    while(count == N) cwait(notfull); // Buffer is full; avoid overflow
11    buffer[nextin] = x;
12    nextin = (nextin + 1) % N;
13    count++;       // One more item in buffer
14    cnotify(notempty);  // Notify any waiting consumer
15  }
16
17  void take (char x){
18    while(count == 0) cwait(notempty); // Buffer is empty; avoid underflow
19    x = buffer[nextout];
20    nextout = (nextout +1) %N;
21    count--;       // One fewer item in buffer
22    cnotify(notfull);   // Notify any waiting producer
23  }
```

Ce moniteur fonctionne avec le producteur et consommateur suivant :

```
1   void producer(){
2     char x;
3     while(true){
4       produce(x);
5       append(x);
6     }
7   }
8
9   void consumer(){
10    char x;
11    while(true){
12      take(x);
13      consume(x);
14    }
15  }
16
17  void main(){
18    parbegin (producer, consumer);
19  }
```

4 Cahier de Spécification

1 Architecture générale du système

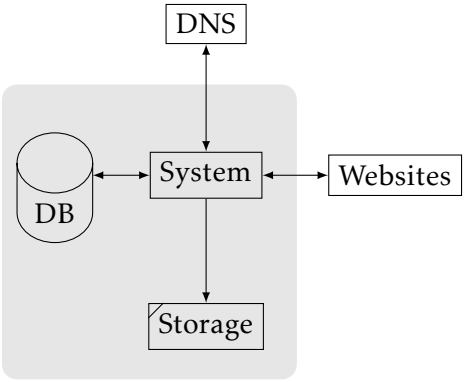


Figure 1 – *Le système dans son environnement*

1.1 Base de données

Une base de données MySql est utilisée pour le stockage de liens. Elle comporte 2 tables :

1. table_url : Table 1.
2. table_file : Table 2.

Table 1 – *Base de données : table_url*

id	id_urlFrom	visited	url	url_hash	dateFound	dateVisited
INTEGER UNSIGNED AUTO_INCREMENT	INTEGER UNSIGNED DEFAULT 0	BOOLEAN DEFAULT 0	VARCHAR(255) DEFAULT NULL	VARCHAR(64) DEFAULT NULL UNIQUE	DATETIME DEFAULT NULL	DATETIME DEFAULT NULL

Table 2 – *Base de données : table_file*

id	id_urlFrom	downloaded	url	url_hash	dateFound	dateDownloaded
INTEGER UNSIGNED AUTO_INCREMENT	INTEGER UNSIGNED DEFAULT 0	BOOLEAN DEFAULT 0	VARCHAR(255) DEFAULT NULL	VARCHAR(64) DEFAULT NULL UNIQUE	DATETIME DEFAULT NULL	DATETIME DEFAULT NULL

2 Description des fonctionnalités

2.1 Fetcher

Le **webcrawler** (ou **Fetcher**) a pour rôle de nourrir la liste des URL sources présentes en BDD. Pour ce faire, il explore un site web (un thread par site web est développé) de manière à en extraire les liens et informations ayant un intérêt pour le projet. Il stocke ces liens dans la base de données en distinguant les liens classiques des liens de téléchargement.

La structure est décrite sur la Figure 2.

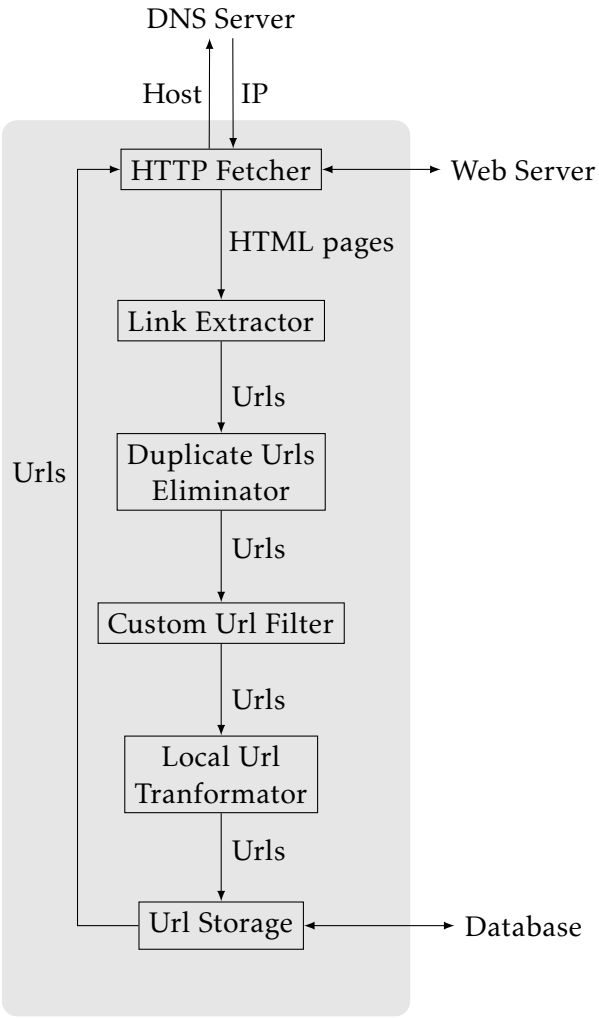


Figure 2 – *Fonctionnement du Webcrawler*

2.2 Downloader

Le *Downloader* est l'outil qui est chargé d'instancier le téléchargement d'un contenu et de la stocker sur le disque dur. Le *Downloader* doit paramétrer le téléchargement entre le client et le serveur de manière à optimiser au mieux le débit.

2.3 Le Scheduler

C'est l'outil chargé de ré-ordonner la file d'attente des moniteurs de manière à sélectionner au mieux le prochain processus à appeler (le ré-ordonnement n'est qu'une ré-évaluation des notes des sites web). Il est appelé à chaque fin d'exécution d'un *Downloader*, les *Fetcheurs* n'étant pas jugés comme pertinents pour l'évaluation d'un serveur.

3 Condition de Fonctionnement

Le système doit être adaptable pour pouvoir fonctionner au mieux sur diverses configurations. Aucune contrainte n'est fixée.

3.1 Performances

Le système doit être plus performant qu'un téléchargement séquentiel des liens. L'objectif est de mettre en évidence que les manipulations réalisées sur les paramètres du protocole de transmission, ainsi que l'ordonnanceur, ont un réel impact sur le téléchargement. La vitesse d'exécution du *Fetcher*, autrement dit la récupération des liens, sera négligée face au temps de téléchargement du contenu à récupérer.

3.2 Capacités

Le système doit être autonome une fois lancé. Pour ce faire, il boucle sans fin à la recherche de nouveaux contenus. Il ne peut être arrêté que par l'utilisateur. Ses capacités ne sont limitées que par le nombre de processeurs et le débit du réseau sur lequel lui et les sites web cibles sont installés.

3.3 Sécurité

Les webcrawlers ont un paramètre de *courtoisie* afin d'éviter de surcharger le serveur qu'ils parcourent et de le soumettre à une attaque DoS [1]. Pour cela le webcrawler devra patienter entre chaque téléchargement et/ou ralentir en cas de non réponse du site web ciblé.

3.4 Intégrité

Le système doit être capable de déterminer (en cas de déconnexion) si le téléchargement en cours a été effectué ou non. Dans le cas où le téléchargement a été interrompu le système doit pouvoir soit le reprendre, soit le recommencer (à déterminer en fonction du scheduler). Le webcrawler n'est pas affecté par les déconnexions et le scheduler non plus, n'ayant pas besoin de ressources externes au système.

4 Gestion du projet

4.1 Planning

Pour ce projet un planning prévisionnel a été réalisé.

1. Attaque de déni de serveur : ou Denial of Service Attack en anglais est une attaque informatique ayant pour but de rendre indisponible un service en l'inondant de requêtes afin de surcharger le serveur cible. Source

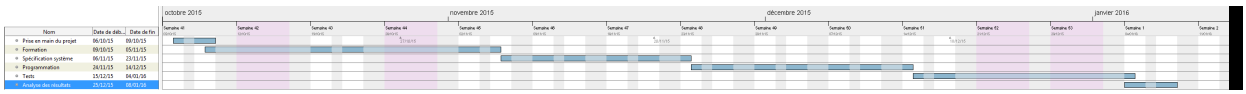


Figure 3 – *Planning prévisionnel*

4.2 Environnement de travail

Le travail est effectué en **Java** à l'aide d'Eclipse Mars.1. Un un dépot Github a été utilisé ainsi que l'outil SourceTree pour la gestion du versionning de code et sa sauvegarde autre que locale. La sauvegarde du contenu téléchargé est effectuée sur un disque dur externe de 7To fournit par l'école.

Les sites attaqués par l'application sont décris dans la section Contexte.

5 Plan de développement

5.1 Découpage du projet en tâches

5.1.1 Etat de l'art

Description de la tâche

L'état de l'art est la première étape, l'objectif est d'en apprendre plus sur le fonctionnement des webcrawlers. Il comporte aussi la recherche de sites web pouvant répondre aux caractéristiques voulues pour le projet. Cette étape relativement courte sert d'introduction au sujet.

Cycle de vie

Cette étude sera réalisée en accord avec les spécifications de l'encadrant du projet, et ces spécifications dépendront donc des remarques et demandes soumises.

Estimation de charge

Cette tâche est estimée à 5 jours/homme en comprenant les recherches liées aux techniques d'ordonnancement pour le téléchargement.

5.1.2 Apprentissage

Description de la tâche

L'objectif de cette tâche est de se mettre à jour sur toutes les technologies qui seront utilisées pour le projet. Cette tâche est effectuée sur l'intégralité du projet étant donné que l'ensemble du projet amène à apprendre sur des sujets divers et variés.

Estimation de charge

Il est estimé que cette tâche prendra 50% du temps du projet.

5.1.3 Création des Fetchers

Description de la tâche

La première étape de programmation est la création des *Fetchers*. Cette tâche est effectuée tôt afin de pouvoir engranger suffisamment de liens pour pouvoir exploiter par la suite le *Downloader* sans manquer de ressources.

Cycle de vie

Le module sera testé sur une journée de travail complète pour savoir s'il rencontre une quelconque erreur sur cette durée qui ne serait pas gérée.

Livrables

Ce module est fonctionnel et autonome. Ne dépendant pas du reste du système, il doit être fonctionnel et livrable, bien que le projet ne nécessite pas de livraison particulière.

Estimation de charge

La charge estimée est à 5 jours/h pour la création du programme puis 1 jour maximum pour la création d'un *Fetcher* spécifique à un site web.

5.1.4 Création du Downloader

Description de la tâche

La seconde étape de développement est la création des *Downloaders*. La particularité d'un *Downloader* est qu'il doit être capable d'instancier le téléchargement de manière à ce qu'il soit le plus optimisé possible, en fonction des paramètres qui lui seront donnés. Pour cela, il interviendra directement sur le protocole TCP afin de définir lui-même les paramètres de connexion avec le serveur. C'est également ce module qui est chargé de l'archivage sur le disque dur.

Cycle de vie

Le module sera testé sur une journée de travail complète pour savoir s'il rencontre une quelconque erreur sur cette durée qui ne serait pas gérée.

Estimation de charge

La charge est estimée est à 4 jours/h pour la création du programme.

5.1.5 Intégration du scheduler

Description de la tâche

La troisième étape de développement est la création des moniteurs et l'intégration de la méthode d'ordonnancement. Cette étape a pour but de synchroniser les processus afin de répondre à la problématique de producteur consommateur.

Cycle de vie

Le module sera testé au moins sur une journée de travail complète pour vérifier si il rencontre une quelconque erreur sur cette durée qui ne serait pas gérée.

Estimation de charge

La charge estimée est à 5 jours/h pour la création du module et son inclusion dans les moniteurs.

Deuxième partie

Développement

5 Méthodologie

Le développement du projet en méthode agile a été nécessaire afin de pouvoir paralléliser l'exécution de tests (dans le but de récolter des données et non pas de vérifier l'intégrité du programme) et l'avancement du développement. Les versions stables ont été mises à jour régulièrement sur Github. Des entretiens réguliers avec l'encadrant ont permis de m'assurer que je restais dans le cadre du projet et de le tenir informé de l'avancement du projet.

6 Mise en œuvre

1 Implémentation

Le projet c'est :
- 33 commits
- 5000+ lignes de code écrites
- 2000+ lignes de code revues
- 3000+ lignes de code final
- 20 classes
- 1 contributeur

1.1 Classes

1.1.1 default package

Launcher

C'est la classe qui doit être configurée dans l'environnement de run. C'est à partir de cette classe que l'utilisateur doit créer la base de données, créer les sites web, le moniteur, ainsi que les *Fetchers* et *Downloaders*. Exemple :

```
1  public class Launcher {
2    public static void main(String[] args) throws Exception {
3      new Database();
4
5      WebSites.getInstance().addWebsite("MangaPanda", "http://www.mangapanda.com/", ↩
               "http://www.mangapanda.com/latest");
6      WebSites.getInstance().addWebsite("DigitalComicMuseum", ↩
               "http://digitalcomicmuseum.com/", ↩
               "http://digitalcomicmuseum.com/stats.php?ACT=latest&start=0&limit=1000", ↩
               "polytech", "p0lyt3ch", ↩
               "http://digitalcomicmuseum.com/forum/index.php?action=login2", "-1");
7
8      Monitor Moniteur = new Monitor();
9      new Crawler_DigitalComicMuseum(Moniteur);
10     new Downloader_Generic(Moniteur);
11   }
12 }
```

1.1.2 concurrency

Le package concurrency contient 2 classes. **Monitor** et **Monitor_Token**. La classe **Monitor_Token** n'est pas finalisée, elle reprend la classe **Monitor** mais adapte son code afin de correspondre strictement à un moniteur et non pas à celui réalisé pour nos besoins spécifiques. Elle doit pouvoir fonctionner avec des ensembles de liens (tokens). N'étant pas finalisée nous n'aborderont que la classe Monitor.

Monitor

Cette classe est la classe de définition de l'objet Monitor qui est un moniteur permettant de coordonner des programmes entre eux. Elle n'est pas définie comme un moniteur standard étant donné un problème rencontré et définit dans la partie Problèmes rencontrés. C'est dans cette classe que l'on peut faire varier le paramètre C qui contrôle le nombre de liens maximum non téléchargés présents en base de données. Il s'instancie avec le constructeur suivant :

```
1  Monitor();
```

SelectIdUrlWebsite

est une classe objet permettant au *Downloader_Generic* de fonctionner. Elle permet de stocker l'url la plus récente en base de données pour chaque site web. Elle a pour vocation à être utilisée lors du fonctionnement d'un Monitor_Token.

1.1.3 data

Database

Cette classe contient les paramètres de la base de données ainsi que les fonctions de créations des tables si elles n'existent pas. C'est également cette classe qui possède la méthode static permettant de transformer un lien en *SHA-256*.

DataSource

DataSource est la pool de connexion à la base de données. Cette classe est écrite pour être un Singleton et ne pas s'instancier plusieurs fois. Elle se base entièrement sur la bibliothèque *c3p0* [1].

Queries

Queries est une classe contenant des fonctions static permettant d'exécuter des requêtes à la base SQL de manière transparente. Il suffit d'utiliser la bonne fonction et de lui passer une requête SQL valide pour obtenir l'effet désiré. Elle contient également 4 fonctions spéciales : *archiveLink(String Link, String Date)*, *archiveFile(String Link, String Date)*, *resetLink(String url, String Date)* et *resetFile(String url, String Date)* qui permettent au *Fetcher* et au *Downloader* de changer l'état d'un lien en base de données à la fin d'un traitement (pour les 2 premiers) ou en cas d'erreur de traitement.

1.1.4 downloader

Downloader_Generic

C'est un *Downloader* fonctionnant avec n'importe quelle url. Il hérite de la classe "*thread.Thread*" et implémente l'interface *Runnable* afin d'être exécuté dans un thread. Il s'instancie avec les fonctions suivantes :

```
1  Downloader_Generic(); //Downloader sans synchronisation inter-thread
2  Downloader_Generic(Monitor monitor); //Downlader fonctionnant avec un moniteur
3  Downloader_Generic(String Name); //Pour donner un nom au Thread
4  Downloader_Generic(String Name, Monitor Moniteur);
```

1. Gestionnaire de connexion MySql en java : http://www.mchange.com/projects/c3p0/

Downloader_Thread

Cette classe est le prédécesseur du *Downloader* générique. Elle fonctionne pour un site web donné uniquement. Elle peut être synchronisée via un moniteur si nécessaire mais ne téléchargera que les liens en provenance du site web passé en paramètre (téléchargement effectué en fonction du nom de domaine du site web, donc si le lien du fichier à télécharger récupéré sur le site web ne possède pas le même nom de domaine le téléchargement ne se fera pas). Elle implémente également l'interface *Runnable* et hérite de la classe "*thread.Thread*".

```
1  Downloader_Thread(WebSite website);
2  Downloader_Thread(WebSite website, Monitor monitor);
3  Downloader_Thread(WebSite website, String Name);
4  Downlaoder_Thread(WebSite website, String Name, Monitor moniteur);
```

Downloader_Url

Il ne télécharge que l'url passée en paramètre et s'arrête. Hérite de la classe "*thread.Thread*" et implémente l'interface *Runnable*. Elle ne peut cependant pas être synchronisée par un moniteur.

```
1  Downloader_Url(String Url);
```

1.1.5 thread

Thread

C'est la classe contenant le nom d'un Thread, le compteur de thread et le moniteur s'il en existe. Toutes les classes implémentés sous l'interface *Runnable* héritent de cette classe dans le projet.

1.1.6 timer

DatabaseCheck

Cette classe contient des fonctions *statics* permettant d'afficher à intervalles réguliers (comme une seconde) le nombre de liens dans la base de données qui sont à télécharger. Elle implémente l'interface *Runnable* et possède un unique constructeur :

```
1 DatabaseCheck(Monitor monitor);
```

TimerCSV

Cette classe contient toutes les fonctions permettant d'écrire des résultats dans un fichier CSV. Elle est très utile pour réaliser les expériences.

```
1 timerToCSV(Date dateStarting, Date dateEnding, int size, String filepath);
2 timerToCSV(Date dateStarting, Date dateEnding, int size, String filepath, int numberThread);
3 timerToCSV(Date dateStarting, Date dateEnding, String comment, String filepath);
4 timerToCSV(String id, String dateFetch, String dateDownloaded, String website, String ↩
        filepath);
```

1.1.7 webCrawler

C'est en fait le package correspondant au Fetcher initialement nommé webCrawler.

Crawler_DigitalComicMuseum

c'est le webcrawler permettant de parcourir le site web DigitalComicMuseum. Il utilise la librairie *Jsoup* pour parcourir un site web puis le parser et en extrait les liens de téléchargement répondant contenant **index.php ?ACT=dl** ou **index.php ?action=dl**. Il implémente évidemment l'interface *Runnable* et hérite de la classe "*thread.Thread*". Il est possible de lui associer un **moniteur** grâce au constructeur adapté.

```
1 Crawler_DigitalComicMuseum();
2 Crawler_DigitalComicMuseum(Monitor moniteur);
```

Crawler_MangaPanda

c'est le fetcher permettant de parcourir le site web MangaPanda. Il implémente l'interface *Runnable* et hérite de la classe "*thread.Thread*". Il parcourt, parse et stocke les liens utiles, les fichiers correspondant aux url contenus dans les balises **<img>**. Il existe deux constructeurs :

```
1 Crawler_MangaPanda();
2 Crawler_MangaPanda(Monitor moniteur);
```

HTTPrequest

C'est la classe contenant les fonctions *static* qui utilisent la librairie Jsoup pour le parcours d'un site web et le parsage. Elle est utilisée par les *Fetcher*.

1.1.8 websites

WebSite

C'est la classe permettant le stockage d'informations concernant un site web. Elle possède également la fonction permettant de se connecter à un site web lorsque la bonne url de connexion est passée en paramètre. Mais aussi deux mutex très importants pour l'accès à la base de données afin qu'un thread ne puisse accéder aux éléments d'un site web dans la base de données seulement lorsqu'aucun autre n'y a déjà accès. Ces mutex sont aux nombres de deux : l'un pour la table des fichiers, l'autre pour la table des url. Il existe 4 constructeurs :

```
1 WebSite(String Name, String Url, ArrayList<String> UrlsPivot);
2 WebSite(String Name, String Url, ArrayList<String> UrlsPivot, String UserName, String ↩
        Password);
3 WebSite(String Name, String Url, String UrlsPivot);
4 WebSite(String Name, String Url, String UrlsPivot, String UserName, String Password, ↩
        String Login_url, String Login_cooke_length);
```

WebSites

Cette classe est un singleton permettant d'accéder aux WebSite créés. Elle initialise les id des sites web et permet ainsi de parcourir l'ensemble des sites web créés par l'utilisateur. Elle est très utile pour l'utilisation d'un moniteur avec *Token*.

1.2 Libraries

Ce projet comprend 4 librairies externes :

- c3p0
- jsoup
- mchange-commons-java
- mysql-connector-java

1.2.1 c3p0

C'est la librairie permettant de gérer une pool de connexion à la base de données. Version 0.9.5.1.

1.2.2 jsoup

C'est la librairie permettant le parsage du HTML pour en extraire des données. Elle peut également requêter des pages HTML en plus de le parser et utiliser des fonctions de connexions à un site web. Elle est utilisée par les *Fetcher* étant donné que les temps de réponse sont sensiblement similaires à une implémentation rigoureuse de socket effectuant le même travail. Version : 1.8.3.

1.2.3 mchange-commons-java

C'est une dépendance de c3p0. Version 0.2.10.

1.2.4 mysql-connector-java

C'est la librairie permettant d'interagir avec une base de données MySql en Java. Version : 5.1.36.

1.3 Base de données

La base de données à été implémentée selon le modèle décrit dans le cahier des charges.

1.4 Fonctionnement

Voir la documentation (JavaDoc) du projet.

1.4.1 Lancement du programme

Comme décrit dans la classe *Launcher* le lancement du programme se fait en exécutant cette classe. Voir figure 1

Il est nécessaire d'initialiser les sites web avant le lancement des thread *Fetcher* et *Downloader*. Une bonne pratique est de commencer par la création de la base de données, puis l'initialisation des sites web, le moniteur et enfin de lancer les threads *Fetcher* et *Downloader*.

```
1 new Database(); //create Database and tables if not exist
2 WebSites.getInstance().addWebsite("MangaPanda", "http://www.mangapanda.com/", ↩
        "http://www.mangapanda.com/latest");
3 Monitor Moniteur = new Monitor();
4 new Crawler_DigitalComicMuseum(Moniteur);
5 new Downloader_Generic(Moniteur);
```

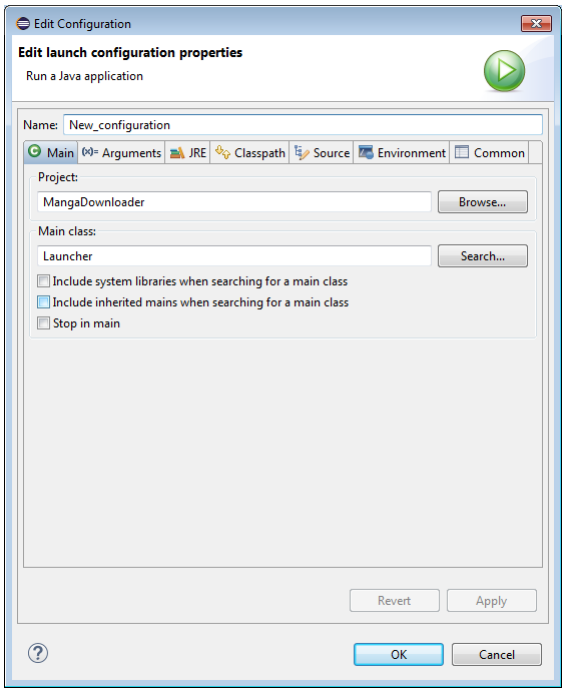


Figure 1 – *Configuration eclipse*

1.4.2 Thread : Fetcher

Les *Fetchers* (nommés webcrawlers dans le programme) doivent être écrits pour chaque site web. Ajouter un site web sans le webcrawler adapté n'aura aucun impact sur le programme et aucune ressource ne sera utilisée de ce site web. L'architecture standard d'un *Fetcher* doit être rédigée comme suit :

```
1  public class Crawler_Standard extends thread.Thread implements Runnable{
2    public Crawler_Standard(){
3      setThreadName("ThreadName");
4      this.start();
5    }
6
7    public Crawler_Stardard(Monitor monitor){
8      setMonitor(monitor);
9      this();
10   }
11
12   private ArrayList<String> getUrlsFromString(String html, WebSite siteweb){
13     //Use jsoup to parse html and return urls for the Fetcher
14   }
15
16   private ArrayList<String> getFilesFromString(String html, WebSite siteweb){
17     //Use Jsoup to parse html and return files url for the the Downloader
18   }
19
20   public void start(){
21     if(getThread() == null){
22       setThread(new Thread(this, getThreadName()));
23       getThread().start();
24     }
25   }
26
27   @Override
28   public void run(){
29     //Step 1 : init website
```

```
30      //Step 2 : login if needed
31      //Step 3 : chose a url from the urlPivot table
32      while(1){
33        //Download html file
34        //Parse it using function described above
35        //Create insert queries based on this Querie : "INSERT INGORE INTO
                database.table_urlToVisit (id_urlFrom, url, url_hash, dateFound) VALUES
                ....." same goes for table_fileToDL
36        //ExecuteQueries
37        //If no errors then archive the link in the database
38        //Else reset the link in the database
39        //Call the monitor if there is one
40        //Select the new url if there is no new url in database use the urlPivot table.
41      }
42    }
43  }
```

1.4.3 Thread : Downloader

Le *Downloader* est générique ou fonctionne avec un site web. Il est codé de la manière suivante :

```
1   public class Downloader_Generic extends thread.Thread implements Runnable{
2     public Downloader_Generic(String ThreadName) throws IOException{
3       setThreadName(ThreadName);
4       this.start();
5     }
6
7     public Downloader_Generic(Monitor monitor) throws IOException{
8       setMonitor(monitor);
9       this("");
10    }
11
12    public void start(){
13      if(getThread() == null){
14        setThread(new Thread(this, getThreadName()));
15        getThread().start();
16      }
17    }
18
19    @Override
20    public void run() {
21      while(1){
22        //Create the socket and set ReceiveBufferSize to use Window Scaling
23        //Call the monitor if there is one
24        //GetNextFileUrl and set the WebSite (the WebSite structure store informations like
                login url, passwords, etc ...)
25        //Login if not already logged and if it needs to be
26        //Initialisation of host and path
27        //Initialisation of the filename based on the url (used for direct file access)
28        // if(filename.isEmpty()) We will have to find the filename later in the http
                header given by the web server
29        //Init folder
30        //Connect the socket and set parametters
31        //Write and send the HTTP request
32        //Read the awnser and check for filename
33        //Download the file
34        //archive the file in the database
35        //if there is any exception reset the file in the database
36      }
```

```
37    }
38 }
```

2 Problèmes rencontrés

Plusieurs problèmes furent rencontrés lors du projet :

Réseau Wifi

pour des raisons pratique le programme fut originellement développé sur mon ordinateur portable plus performant que les machines virtuelles de l'école. Les tests fonctionnels ne fonctionnaient pas dans un premier temps à cause des restrictions Wifi imposés par l'école. L'utilisation d'un VPN a permis de contourner ces restrictions et de pouvoir effectuer les tests fonctionnels sur l'ordinateur portable tout en effectuant les campagnes de tests sur la machine virtuelle (elle connectée au réseau filaire).

Gestion de la base de données

la gestion de la base de données fut refaite à plusieurs reprises et donna lieu à beaucoup de refontes du code du projet. C'est un aspect critique du projet car les requêtes doivent être rapides et efficaces. Elle donna notamment lieu à une question sur stack-overflow [2] pour optimiser son implémentation.

2.1 Moniteur non conventionnel

Le problème majeur rencontré lors du projet vient de l'implémentation du moniteur. En Java il existe une classe *BlockingQueue* qui implémente un moniteur fonctionnel. L'implémentation du moniteur reprend donc le code de *BlockingQueue* décrit ici et l'adapte pour notre cas. La spécificité du programme provient du fait que la base de données gère toute seule les doublons (avec une clé unique positionnée sur le *SHA-256* d'une url). Ainsi la requête SQL d'ajout dans la base de données ignore les doublons. Et ce n'est donc qu'après avoir ajoutées tous les liens à la base de données que l'on peut savoir combien de liens ont réellement été ajouté. Le moniteur ne peut donc pas être appelé à chaque génération de lien unique (on ajoute pas les liens un par un mais par blocs pour des questions de performances). Il faut donc créer un moniteur qui puisse prendre en compte l'ajout multiple de ressources à la base de données. Pour ce faire une première étape fut d'adapter trivialement le code du moniteur pour ajouter plus d'un élément à la fois :

```
1  public void append (int nbr) throws InterruptedException{
2      lock.lock();
3      try{
4        while (count >= C) //C is the max allowed  ressource
5          notfull.await();
6        count+= nbr;
7        notempty.signal();
8      } finally{
9        lock.unlock();
10     }
11   }
```

Mais le problème est que cette méthode ne tient pas à jour la valeur *count* comme elle le devrait. On observe un décalage entre la valeur *count* et le nombre réel de liens en attente d'être téléchargés dans la BDD. Étant donné que nous ajoutons dans tous les cas les liens pour savoir le nombre réel ajouté, l'accès au moniteur ne protège pas véritablement l'écriture en base de données. La première étape dans le moniteur doit donc être la mise à jour de *count* avant d'arrêter le thread en cours (l'ajout ayant déjà été effectué de toute manière). Le code final est donc :

2. JDBC optimize MySql request on Multithread : http://stackoverflow.com/questions/34743721/jdbc-optimize-mysql-request-on-multithread

```java
public void append (int nbr) throws InterruptedException{
    lock.lock();
    try{
      count+= nbr; //Since i'm adding into the database before requesting the lock for efficiency purpose I need to keep count update with the database before locking the thread
      while (count >= C){
          notfull.await();
      }
      notempty.signal();
    } finally{
      lock.unlock();
    }
  }
```

Avec une adaptation de la fonction *take()* qui n'appelle plus un producteur à chaque fois qu'une ressource est récupérée. Mais seulement si *count* < C.

```java
public void take() throws InterruptedException{
    lock.lock();
    try{
      while(count == 0){
        notempty.await();
      }
      count--;
      if(count < C)
        notfull.signal();
    }finally{
      lock.unlock();
    }
  }
```

Cette solution casse l'aspect strict du moniteur et permet donc dans certains cas de dépasser le nombre maximum fixé noté *C*.

3 Planning

Le planning effectif est légèrement différent du planning prévisionnel. Tout d'abord une tâche à été ajoutée : la recherche, qui n'était pas différenciée de la formation dans le planning prévisionnel. Il y a aussi les étapes de programmation, tests et analyse des résultats qui ont été effectuées en parallèle. Ce changement provient de la durée des tests (de 15 minutes à 24 heures). Les tests ont donc été lancés plus tôt que prévu et la programmation effectué en parallèle de leurs exécution.

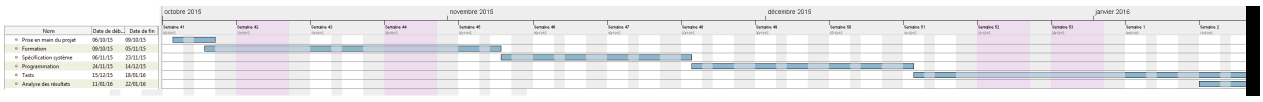


Figure 2 – *Diagramme de Gantt effectué*

7 Campagne de tests

Les tests ont été effectués dans 2 buts : l'évaluation des performances et l'évaluation de la synchronisation.

1 Performances

1.1 Évolution de la vitesse de téléchargement

Afin que l'implémentation de la prédiction temporelle ait du sens, il était important d'observer une variation dans les temps de réponses ou de téléchargement dans le temps. Pour vérifier si de telles variations existent sur les sites web ciblés des téléchargements ont été lancés sur 24 heures. Le résultat de ces tests sont disponibles dans le fichier : Download_DigitalComicMuseum_NoWrite.xlsx pour DigitalComicMuseum et Download_MangaPanda_By_Serv.xlsx pour MangaPanda. Tous les fichiers dont le nom commence par *Download* correspondent à cet expérimentation. Il faut savoir que pour mangapanda par exemple l'expérience fut compliquée par la présence de plusieurs serveurs distingués par le nom de domaine : i1.mangapanda.com, i2.mangapanda.com, i3, i4, ... (voir Figure **??**.

1.1.1 Analyse

Les courbes Figure 1 et Figure 2 démontrent bien qu'il n'y a pas de variation que ce soit pour MangaPanda [1] ou pour DigitalComicMuseum. Nous n'observons aucune diminution de la vitesse de téléchargement dans le temps pour une période donnée. Les services répondent donc de la même manière à toute heure de la journée quel que soit le nombre de visiteurs présents sur le site. Ce sont des sites très stables. L'implémentation de la prédiction temporelle est donc remise en question d'autant plus que nous venons à douter de son efficacité. N'ayant qu'une seule source d'information à ce sujet nous ne savons pas si cette source est complète. Il fut donc décidé de relayer cet aspect du projet au second plan.

1. La diminution d'activité observable vers 16h48 sur le graphique ne provient pas de mangapanda.com mais bien de notre système. Les téléchargements ont été lancés sur l'intégralité des liens précédemment récupérés par le *Fetcher*. Il s'est avéré que globalement pour MangaPanda la vitesse du *Downloader* est légèrement plus rapide que celle du *Fetcher*. La raison de cette diminution du taux de téléchargement provient donc du fait que la base de données n'était plus suffisamment alimentée en fichiers à télécharger pour que le *Downloader* s'exécute sans interruption.

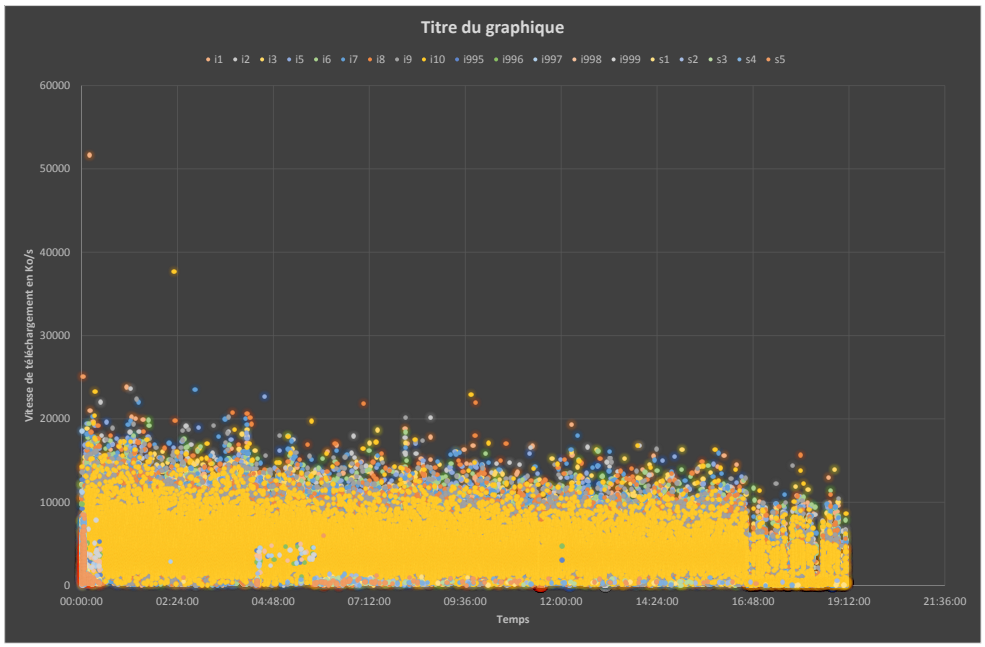


Figure 1 – *Evolution de la vitesse de téléchargement sur MangaPanda trié par serveur*

1.2 Vitesse d'accès à la base de données

Afin de vérifier que le temps d'accès à la base de données est négligeable il faut avoir un rapport d'au moins 100 entre le temps total d'exécution du processus et le temps total d'accès à la base de données. Pour comparer les temps d'accès à la base de données des *Fetchers* et *Downloaders* ont été enregistrés ainsi que leurs temps d'exécution. Les résultats se trouvent dans les fichiers : Timer_Threads_Size_10.xlsx et Timer_Threads_Size_100.xlsx. Où comme le nom l'indique le nombre de *Downloaders* et de *Fetchers* fut respectivement de 10 et 100.

Table 1 – *Moyenne des temps processus en ms pour N = 10*

	Downloader MangaPanda	Downloader DigitalComicMuseum	Fetcher MangaPanda	Fetcher DigitalComicMuseum
Thread	53.3	46215.2	287.33	872.73
Database	4.7	9.4	38.53	44.73
Parsing	N/A	N/A	150.93	61.27

Table 2 – *Moyenne des temps processus en ms pour N = 100*

	Downloader MangaPanda	Downloader DigitalComicMuseum	Fetcher MangaPanda	Fetcher DigitalComicMuseum
Thread	52.88	47013.72	127.64	563.05
Database	5.3	15.12	19.2	36.41
Parsing	N/A	N/A	33.57	15.69

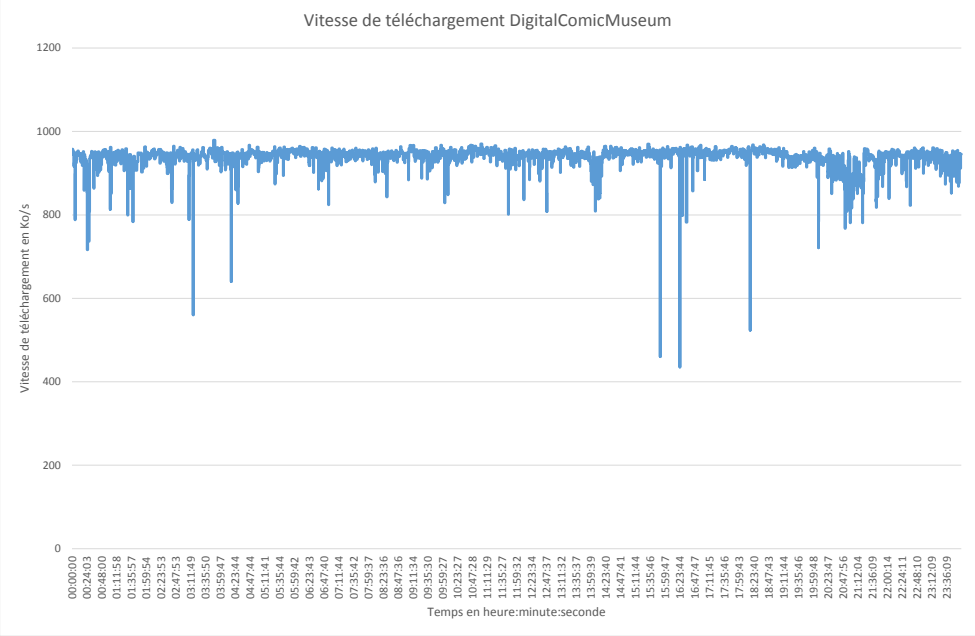


Figure 2 – *Evolution de la vitesse de téléchargement sur DigitalComicMuseum*

1.2.1 Analyse

Les résultats montrent que le temps moyen est supérieur à 100. Mais si nous observons les temps en fonction des sites web, dans le cas de mangapanda la taille des fichiers à télécharger étant très petite (environs 50Ko) le temps d'accès à la base de données n'est pas négligeable. Nous considérons donc les résultats dans leurs ensemble et estimons que le temps d'accès à la base de données est négligeable.

1.3 Window Scaling

Un simple test de téléchargement d'un fichier de 50Mo avec et sans *window scaling* à été réalisé.

Table 3 – *Test du Window-Scaling sur un fichier de 55Mo*

	Sans Window Scaling	Avec Window Scaling
Buffer Size	32Ko	128Ko
Temps mm :ss	04 :01	02 :11

1.3.1 Analyse

Nous venons de démontrer que l'utilisation du *window scaling* augmente bien la vitesse de téléchargement et donc permet d'optimiser l'utilisation de la bande passante.

1.4 Vitesse de téléchargement

Une expérience importante fut d'augmenter le nombre de thread et de voir l'impact sur la vitesse de téléchargement. Afin de s'assurer que le nombre de thread permet bien d'augmenter l'efficacité du programme. Les résultats sont dans le fichier Vitesse_Nbr_Thread.xlsx.

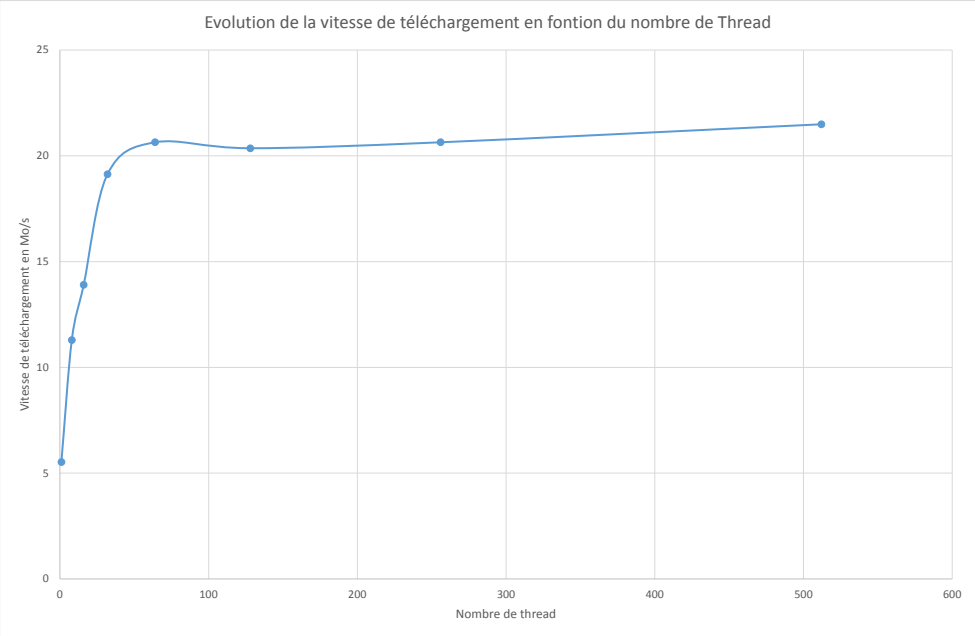


Figure 3 – *Vitesse de téléchargement globale en fonction du nombre de Thread*

1.4.1 Analyse

Les résultats obtenus démontrent bien que l'augmentation du nombre de thread permet d'optimiser l'utilisation de la bande passante. Par contre il permet également de mettre en avant qu'a partir de 64 *Threads* l'utilisation de la bande passante n'augmente plus significativement.

2 Synchronisation

2.1 État de la base de données dans le temps

La vérification du bon fonctionnement de la synchronisation et l'impact du moniteur passe par le monitoring du nombre de liens en attente d'être téléchargé dans la table *table_file* de la base de données. Ce monitoring est effectué à intervalle de temps régulier : 1 seconde. En prenant une capacité du moniteur (noté C) à 10. Les résultats sont enregistrés dans le fichier *Database_State.xlsx*

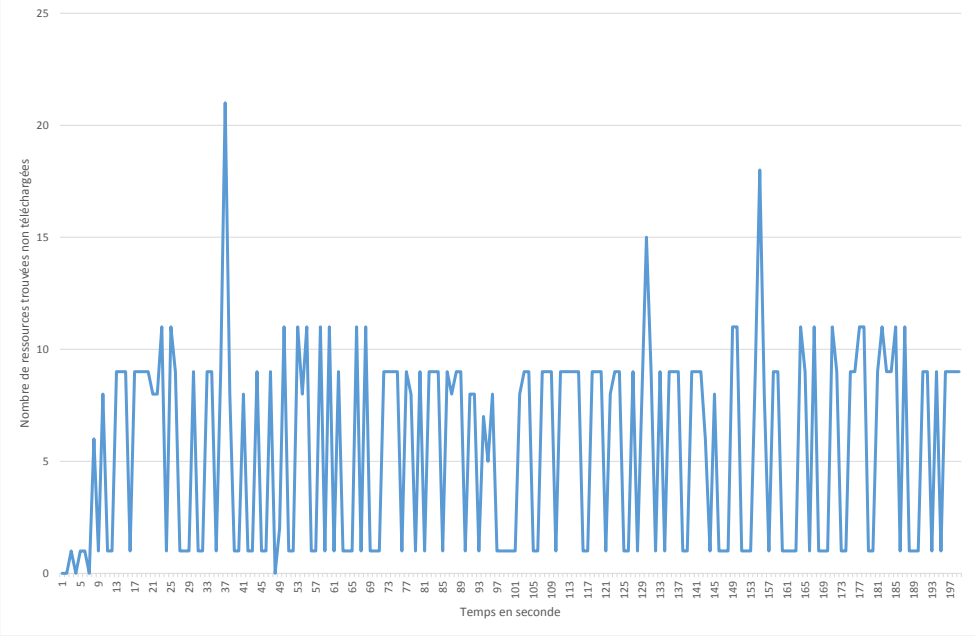


Figure 4 – *Nombre de ressources non téléchargées dans la base de données*

2.1.1 Analyse

On constate que le système fonctionne bien et limite le nombre de liens en attente de téléchargement dans la base de données. Comme décrit précédemment le moniteur n'est cependant pas parfait et certains pics dépassant le nombre maximum autorisé sont constatés. Ces pics sont systématiquement suivit par une diminution dans la base de données démontrant bien que l'intégralité des *Fetchers* sont stoppés tant que le nombre de ressources n'est pas repassé en dessous de 10.

2.2 Détermination du temps de synchronisation

Le temps de synchronisation est le temps entre le moment où une ressource à été déposée par un *Fetcher* dans la base de données et le moment où le *Downloader* vient la récupérer. Ce temps est important car l'environnement web étant pervasif, avec des migrations de ressources, ou suppressions rien ne garantit la pérennité d'un lien dans le temps. L'objectif est donc de contenir ce temps. Notre solution de producteur consommateur permet de contenir ce temps. Nous l'avons monitoré pour deux tailles de fichiers en fonction du nombre de thread : le téléchargement de fichiers de 4Mo et celui de 50Ko. Voir Figure 5.

2.2.1 Analyse

Nous constatons un résultat simple : l'augmentation du temps de synchronisation en fonction du nombre de thread est linéaire. La courbe possède une échelle logarithmique pour des questions d'esthétismes mais aucun doute sur la linéarité des résultats. Un autre constat réalisé est que lorsque le fichier augmente de taille le temps de synchronisation aussi. Il est donc très important de bien fixer le nombre de thread en fonction de la cible afin de contenir le temps de synchronisation.

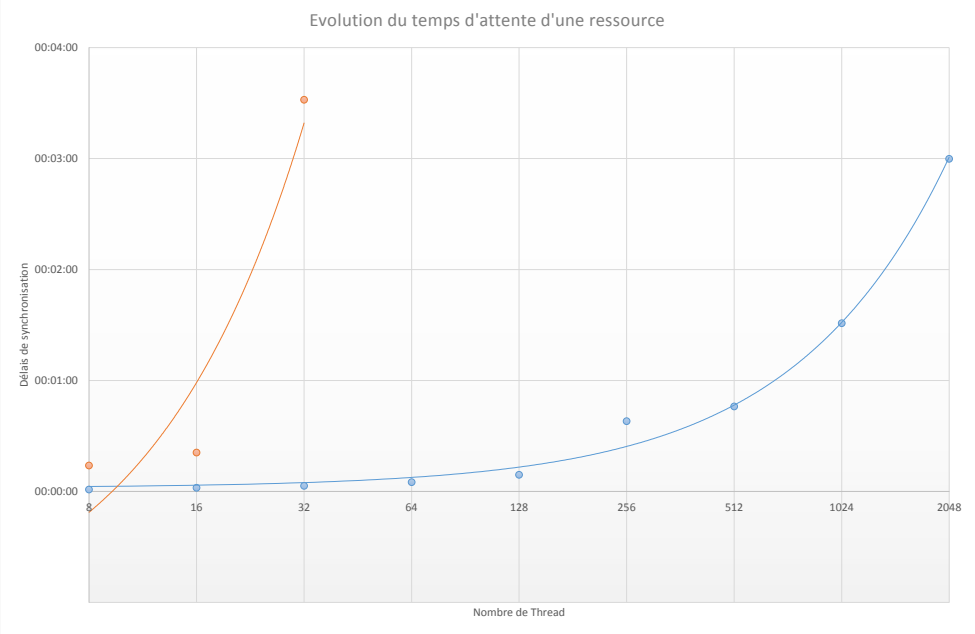


Figure 5 – *Évolution du délais de synchronisation*

8 Améliorations

Au cours du projet certaines améliorations du programme ont été envisagées sans pouvoir les mettre en œuvres faute de temps.

1 Gestion des socket

Il est possible d'améliorer grandement les performances du programme en utilisant un pool de socket. C'est à dire que tout comme un pool de connections pour la base de données, un pool de socket permettrait de stocker des socket initialisés mais non utilisés (avec un timeout inexpiré). Ainsi lorsqu'on associe la prochaine cible de téléchargement d'un *Downloader_Generic* celui ci pourrait aller chercher dans la pool de socket un socket pointant vers le site cible déjà créé par un précédent *Downloader* sans avoir à recréer un nouveau socket.

2 Reprise d'un téléchargement interrompu

Il devrait être possible de gérer la reprise d'un téléchargement interrompu avec 2 méthodes différentes :

2.1 Reprise HTTP

En cas de téléchargement interrompu dans certains cas il est possible de demander au serveur de reprendre le téléchargement. Dans ce cas la communication reprendra là où elle s'était arrêté précédemment.

2.2 Reprise d'écriture

Si la première solution n'est pas envisageable, il faut alors relancer le téléchargement. Afin de gagner du temps il est possible de ne pas écrire les données déjà présentes dans le fichier non complet. Strictement parlant le téléchargement reprendra depuis le début mais une économie sera faite au niveau du temps d'écriture.

3 Finalisation du projet

3.1 Tokens

Comme amélioration possible (ne prenant pas en compte les performances) la finalisation du projet est envisageable. Avec notamment l'implémentation des classes *Downloader_Generic_Token* et Monitor_Token. Le but étant que le moniteur retourne au *Downloader* un site web lorsque celui ci à atteint un nombre de téléchargement à exécuter en une fois. Par exemple télécharger les images 20 par 20 et non pas à l'unité. 20 images correspondent alors à un *Token* et le moniteur peut alors reprendre un aspect stricte et fonctionnel comme décrit dans la documentation java. Il est même envisageable d'utiliser la structure *BlockingQueue*. On incrémente la valeur d'un site web dans moniteur lorsque le *Fetcheur* à récupéré le nombre d'éléments définissant la taille d'un *Token* pour ce site web.

3.2 Prédiction Temporelle

Un autre aspect de continuité de projet envisageable c'est la prédiction temporelle qui fonctionnerait avec la notion de *Token*. Comme décrite en première partie elle permettrait de choisir un site web pour lequel lancer un téléchargement non pas selon la date d'entrée en base de données mais selon les temps de réponse des sites web. Cet aspect à été minimisé au cours du projet pour une simple raison : les sites web sont stables. Cette prédiction temporelle est très utile lorsque le temps de réponse d'un site web évolue dans le temps. Or les cibles choisis dans le projet sont des sites web dont le temps de réponse n'évolue pas ou très peu dans le temps. La source concernant la prédiction temporelle étant unique nous avons remis en question son utilité au cours du projet et minimisé son importance au profit de l'implémentation de la synchronisation et des expérimentations.

Conclusion

Ce projet m'a permis de monter en puissance sur la programmation réseau en Java mais également de mettre en application des notions vues lors de mon apprentissage à Polytech Tours. J'ai été autonome sur la gestion du projet, la compréhension du sujet, le développement du programme. J'ai pu créer mon propre WebCrawler fonctionnel permettant de récupérer du contenu images de sites web ciblés.

Ce projet n'est néanmoins pas abouti, des améliorations restent possibles. Une suite doit être donnée afin d'améliorer le programme, de le compléter et surtout de faire de plus amples tests vis à vis des performances ainsi que de l'utilité de la prédiction temporelle.

Annexes

Comptes rendus hebdomadaires

Compte rendu n°1 du 09/10/2015

Cette semaine il a été décidé que :

- Le projet ne devait pas rendre compte d'un *web crawler* générique mais d'un *web crawler* spécialisé dans l'extraction de données définies sur des sites web ciblés préalablement.
- Le projet s'effectuerait à raison de 4 jours par semaines avec absence le mercredi sauf en cas d'exceptions.
- Le web crawler lors de la récupération d'une archive devra en plus référencer des informations concernant l'auteur de l'œuvre, la date de sortie et autres méta-données.
- Ne pas se focaliser sur l'industrie du manga ou de la bande dessinée.
- La base de données collectée ne serait pas rendue publique et je dois considérer qu'une fois téléchargée la ressource est consommée. Je n'ai donc pas à me soucier du critère légal des œuvres qui pourront être ciblé par le système.
- Les 3 sites choisis pour réaliser les tests sont :
  - The Digital Comic Museum : http ://digitalcomicmuseum.com
  - ComicBook+ : http ://comicbookplus.com
  - Mangapanda : http ://mangapanda.com

Compte rendu n°2 du 23/10/2015

Cette semaine j'ai préparé le projet :

- Création d'un dépot privé Git sur github
- Installation de Java, Eclipse, Git et Source Tree, MySql Server, MySql Connector J, MySql Workbench.
- Formation sur les protocoles et bibliothèques Java pour le webcrawler.

J'ai également commencé à coder un webcrawler pour le site `www.mangapanda.com` afin de comparer les différentes méthodes de récupération d'une url. L'utilisation de la bibliothème *Jsoup* pour le parsage HTML permet de récupérer une page HTML dans des temps égaux voire globalement inférieure à la programmation par socket. Le webcrawler n'étant pas l'étape à optimiser (mais plus le scheduler et le downloader) il sera développé à l'aide de cette bibliothèque pratique.

Edit du 23/10 : la base de données a été changée pour plus de praticité. Le webcrawler est désormais fonctionnel pour le site mangapanda.

Compte rendu n°3 du 06/11/2015

La connexion au site web est gérée par la librairie Jsoup. Les méthodes de parcours de site web gèrent désormais les cookies afin de rester connecté. La structure site web a été adaptée en conséquence. Afin d'avancer plus vite dans le travail, le nombre de sites a été réduit de 3 à 2. Les deux sites étant : comicbookplus et mangapanda la différence de ces sites permettant d'avoir des tailles variées de fichiers à télécharger. Ceci laisse la possibilité d'avoir des cas intéressants pour tester le scheduler.

Compte rendu n°4 du 13/11/2015

Le webcrawler est fonctionnel. Deux threads tournent sur les sites web : digitalcomicmuseum et mangapana. Digitalcomicmuseum a été priorisé face à comicbookplus pour une question de simplicité de connexion au site web (nécessaire afin d'avoir les liens). Beaucoup de travail sur l'état de l'art et sur la formation java pour la compréhension de l'implémentation réseau avec le livre [5].

Compte rendu n°5 du 20/11/2015

Correction de quelques bugs du programme. Le cahier de spécification ainsi que le rapport ont été avancés. Le fonctionnement de webcrawler est complètement assimilé. La partie ordonnancement également, il faut néanmoins éclaircir la prédiction temporelle.

Compte rendu n°6 du 27/11/2015

Après discussion avec Mr Aupetit il est devenu clair que le cahier de spécification et le rapport ne devaient faire qu'un seul document contrairement à l'année dernière. Je reprends la partie recherche sur les solutions pour le problème producteur consommateur que nous avons échangé avec la partie des moniteurs.

Compte rendu n°7 du 04/12/2015

Les parties optimisation réseau avec le windows scaling et méthodes d'ordonnancement et de fonctionnement des webcrawlers sont complètement acquises. J'ai un peu plus de mal à comprendre les moniteurs, non pas leur fonctionnement mais la manière dont je pourrais les inclure dans le projet pour qu'ils fonctionnent comme je le désire. Le rapport n'a pas avancé il est toujours dissocié du cahier de spécification. La date de ma soutenance de mi-parcours est fixée au 10/12/2015 à 15h en salle 110. Comme précisé ensemble je me concentre sur la préparation de cette présentation au détriment du rapport qui, dans un premier temps sera plus léger que sa version finale. L'objectif étant que j'assimile correctement les principes fondamentaux des moniteurs et leurs fonctionnement et que je prépare le support de la présentation.

Compte rendu n°8 du 11/12/2015

Cette semaine a été dédiée en grande partie à ma soutenance et sa préparation ainsi qu'à la rédaction du rapport. Une montée très forte en compétence sur les moniteurs a été réalisée.

Compte rendu n°9 du 18/12/2015

Cette semaine nous avons décidé de mettre en place des objectifs de tests et résultats au vue de l'avancement du projet dans son développement. Le Downloader est générique et fonctionnel. Il intègre parfaitement les problématiques de login que l'on retrouve sur un des sites tests. Il est donc convenu de réaliser au plus tôt un monitoring des temps de téléchargement sur une longue période (à définir mais je pense faire les tests sur une période de 24h) avec différent scénarios :

1. Un thread de téléchargement unique. (permettant de voir le temps de réponse du serveur)
2. Evaluation de la prévision temporelle.
3. Evaluation de la parallélisation.
4. Le projet finit

Les trois premières étapes permettront d'avoir des résultats même si le projet final n'aboutit pas complètement faute de temps.

Compte rendu n°10 du 08/01/2016

Cette semaine nous avons remis en question la prediction temporelle, nous l'avons relégué au second plan en attente des résultats de monitoring de l'évolution de la vitesse de téléchargement dans le temps d'un site web. Après expérimentation il est apparu que les ressources du site web MangaPanda se trouvent sur différents sites web. De nouvelles expérimentations sont lancées de manière à différencier les temps de réponse des différents serveurs en se basant sur le nom de domaine.

Compte rendu n°11 du 15/01/2016

Cette semaine le moniteur à été codé et est fonctionnel. L'expérimentation sur mangapanda ne mets pas en avant des vitesses différentes selon le serveur ciblé. La vitesse globale de mangapanda est donc stable sur une durée de 24H comme DigitalComicMuseum. L'intégration de la prédiction temporelle n'est donc plus une étape clé du projet et est relayée au second plan.

Compte rendu n°12 du 22/01/2016

Cette semaine la gestion de la BDD a été revue selon les conseils de personnes compétentes à ce sujet. Un Downloader générique a également été implémenté pour pouvoir continuer les tests de manière globale et non pas en ciblant les sites web comme précédemment. Cette étape est aussi une étape clé pour la création du downloader final du projet (comme souhaité initialement) qui puisse fonctionner pour n'importe quel site web et téléchargé le nombre de ressources définis dans un Token du site. Des expérimentations sur l'évolution du nombre de ressources non téléchargées dans la base de données en fonction du temps ont été réalisés en faisant variés les paramètres. Mais également l'évolution de la bande passante en fonction du nombre de thread. Cette dernière a été effectuée sur des sites web permettant le téléchargement de fichiers de plus ou moins grande taille dans le but spécifique de tester la bande passante.

A noter que l'exploitation des résultats de monitoring des serveurs sur 24h donnant un nombre important d'entrée est rendue difficile par le logiciel Excel (fichier très lourd).

Compte rendu n°13 du 29/01/2016

Cette semaine les tests ont été avancé. J'ai également continuer la rédaction du rapport et préparer ma soutenance.

Webographie

[WWW1] Carlos Castillo. *Scheduling Algorithms for Web Crawling*. URL : chato.cl/papers/crawling_thesis/scheduling.pdf (visité le 19/11/2015).

[WWW2] *Hypertext Transfer Protocol*. URL : https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol (visité le 23/10/2015).

[WWW3] *Page Rank*. URL : https://fr.wikipedia.org/wiki/PageRank (visité le 18/11/2015)

[WWW4] *Robot d indexation*. URL : https://fr.wikipedia.org/wiki/Robot_d'indexation (visité le 11/09/2015).

[WWW5] *StackOverflow*. URL : http://stackoverflow.com/.

[WWW6] *Uniform Resource Locator*. URL : https://fr.wikipedia.org/wiki/Uniform_Resource_Locator (visité le 02/09/2015).

[WWW7] *What is a CBZ File ?* URL : http://pcsupport.about.com/od/fileextensions/f/cbzfile.htm (visité le 11/11/2015).

Bibliographie

[1] Anthony DEMARCY et Florian MONTALBANO. *Développement d'un logiciel de téléchargement parallélisé pour la protection du copyright des mangas*. Juin 2015.

[2] Shigeo Sugimoto. Jane Hunter. Andreas Rauber. Atsuyuki Morishima (Eds.) *Digital Libraries : Achievements, Challenges and Opportunities*. Sous la dir. de Springer. 9th International Conference on Asian Digital Libraries ICADL 2006 Kyoto Japan November 2006 Proceedings. Springer, nov. 2006, p. 415–417.

[3] Jaswinder Pal Singh Fengyun Dongming. « Scheduling Web Crawl for Better Performance and Quality ». Princeton, NJ 08540, USA. Thèse de doct. Princeton, 2003.

[4] Christopher Olston et Marc Najork. « Web Crawling ». In : *Foundations and Trends® in Information Retrieval*. T. 4 No. 3. Now publisher, 2010, p. 175–246.

[5] Esmond Pitt. *Fundamental Networking In Java*. Springer Science Business Media, 2010.

[6] William Stallings. *Operating Systems Internals and Design Principles*. Edinburgh Gate, Harlow, Essex SM20 2JE, England : Pearson Education Limited, 2012, p. 246–252. isbn : 0-273-75150-6.

[7] W. Richard Stevens. *TCP/IP Illustrated*. T. 1. Addison Wesly, 2000. isbn : 0-201-63346-9.

Projet Recherche & Développement

Développement d'un web crawler pour la protection des mangas sous copyright

Aurélien PLANCHON

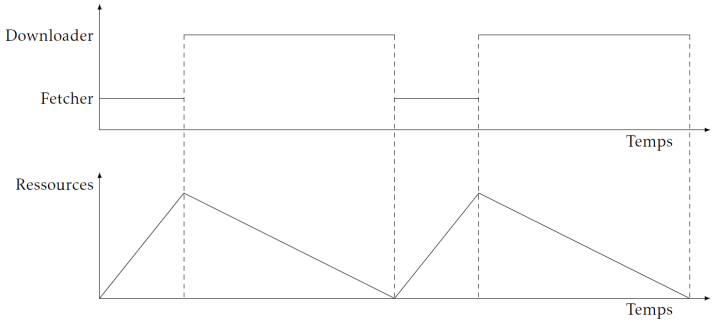
Encadrement : Mathieu Delalandre

Objectif

Réaliser un programme téléchargeant automatiquement des images depuis des sites web ciblés. Le projet s'inclue dans un processus d'analyse d'image visant à détecter des œuvres sous copyright pour lutter contre la fraude dans l'industrie du manga.

Mise en œuvre

Le projet se base sur l'implémentation de 2 modules. Une des étapes clés est la **synchronisation** des deux modules pour répondre au problème producteur/consommateur. Le **Fetcheur** récupère les liens des images (producteur de ressources) pour alimenter le **Donwnloader** (consommateur) qui les télécharge. Dans ce projet chaque module est implémenté en thread pour pouvoir en lancer plusieurs simultanément.



Alternance Fetcher-Downloader

Résultats attendus

- Programme autonome.
- Optimisation de la bande passante.
- Ordonnancement des threads.
- Constitution d'une base d'image.
- **Téléchargement des images plus rapide qu'une simple queue FIFO**

*Couverture du **Jump** 2308 publié le 13 Aout 2015*

Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
www.polytech.univ-tours.fr

POLYTECH TOURS
Informatique

Développement d'un web crawler pour la protection des mangas sous copyright

Résumé

Ce document est le rapport de projet de fin d'étude portant sur l'implémentation d'un webcrawler. L'objectif de ce projet fut de réaliser un webcrawler personnalisé afin de télécharger de manière optimisée des images de sites web ciblés. Le programme est effectué en Java et répond à une problématique de producteur/consommateur avec un moniteur. Un ordonnancement adapté avec une prédiction temporelle est mis en place afin d'optimiser le débit de la bande passante à disposition.

Mots-clés

Webcrawler Moniteur Producteur Consommateur Ordonnancement Java Multithread TCP/IP

Abstract

This document is the report of an end of studies' project, dealing with an implementation of a web crawler. The goal was to implement a specific webcrawler design to download images on targeted websites. The software is implemented with Java language and answer to a producer/consumer problem with monitor. A scheduling with temporal prediction is set up in order to optimise the given bandwidth.

Keywords

Webcrawler Monitor Producer Consumer Scheduling Java Multithread TCP/IP

Tuteurs académiques
Mathieu Delalandre

Étudiants
Aurélien PLANCHON (DI5)