RenPar'15 / CFSE'3 / SympAAA'2003
La Colle sur Loup, France, 15 au 17 octobre 2003

Us : Prototype de Stockage Pair à Pair*

Olivier Soyez

LaRIA,
5 rue du moulin neuf,
80000 AMIENS - France
soyez@laria.u-picardie.fr

Résumé
Les systèmes de stockage pair à pair reposent sur des nœuds très volatiles : le temps de connexion d'un ordinateur sur Internet n'est que de quelques heures. Aussi, la pérennité des données est-elle difficile à garantir. Pour cela, les systèmes existants se basent sur des mécanismes de redondance des données. Suite à une analyse quantitative de ces mécanismes, nous avons réalisé un premier prototype de stockage pair à pair. Il est présenté dans cet article.

Mots-clés : pair à pair, stockage, redondance, fiabilité, distribué

---

1. Introduction

Aujourd'hui les systèmes de type pair à pair sont très largement utilisés pour partager des ressources à travers l'Internet. Certains ont pour but de partager du temps CPU : Seti@Home, XtremWeb, d'autres de mettre des données à disposition d'autres utilisateurs : Gnutella, Freenet. Parallèlement, des systèmes pair à pair dédiés au partage d'espace disque ont vu le jour tel Intermemory, Oceanstore, PAST. Leur but premier est d'offrir un accès transparent à un service de stockage distribué. Les systèmes de partage de CPU peuvent faire face à la disparition d'un nœud en relançant un calcul sur un autre nœud. Pour un système de stockage, la défaillance d'un nœud implique la perte définitive des données qu'il stockait. Cela rend leur réalisation encore plus délicate.
Pour expérimenter les différents mécanismes de stockage pérenne pair à pair, nous développons actuellement une nouvelle plate-forme dans le cadre de l'ACI GRID CGP2P. Cette plate-forme doit nous permettre d'étudier l'efficacité des mécanismes de redondance, les techniques de detection de défaillance, les méthodes de distribution des données, les coûts de maintien de l'intégrité du système.
Dans cet article, nous présentons le premier prototype de cette plate-forme. Nous expliquons son interface utilisateur puis son architecture. Enfin, nous présentons trois modes de déploiement.

2. Interface

Us, Underground Storage [2], est un outil qui permet de stocker des données sur Internet. Il se compose de deux entités principales : un **client** qui est un demandeur d'espace et un **fournisseur** de service qui alloue et gère l'espace. Us est un système de stockage bas niveaux, il peut être vu comme un disque dur virtuel. Ainsi il ne fournit pas de gestions complexes tels que gestion des méta-données, cohérence des données, gestion des droits d'accès. En résumé, Us fournit un espace de stockage dans lequel on peut stocker, lire et effacer des *blocs* de données. Il peut être considéré comme une brique de base d'un système de fichiers ou peut être utilisé pour les accès aux données de programmes. Les utilisateurs de Us peuvent utiliser une interface similaire aux clients FTP ou utiliser un pilote de disque dur. L'objectif principal de Us est le stockage pérenne de données brutes. Pour des raisons de portabilité, Us est écrit en Java et les communications sont gérées par RMI. Us est inter-opérable avec XtremWeb [3] et Jxta [1].

* Us (http://www.ustorage.net) est supporté par l'ACI GRID CGP2P du MENRT

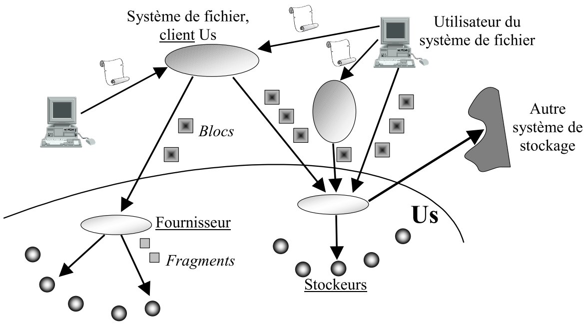

FIG. 1 – Architecture globale de Us

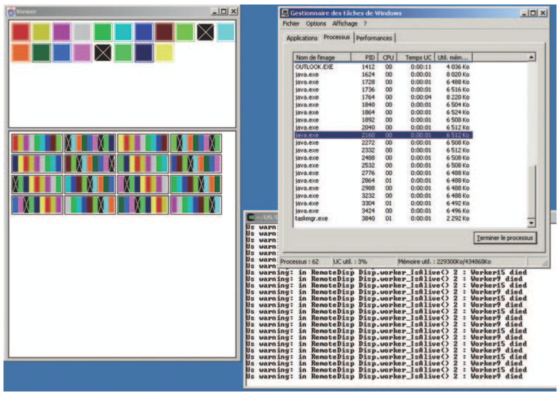

FIG. 2 – Les processus morts (en haut) sont représentés par une croix et induisent la perte de fragments (en bas) représentée pareillement. Lors de démonstrations, les processus sont tués par le gestionnaire d'applications du système.

L'objectif de l'interface utilisateur de Us est de fournir un service Internet sous forme de requêtes, tout en restant proche de l'interface standard d'un disque dur. Pour stocker et gérer ses données, l'utilisateur de Us dispose de deux classes : **Broker**, pour la mise en relation du client avec un fournisseur de service distant ; **Space**, pour représenter l'espace de stockage virtuel. Cet objet est composé d'un ensemble de blocs de taille fixe.

Le Broker ne dispose que de deux méthodes. Les données nécessaires à sa construction sont écrites dans un fichier de configuration à l'installation. Les méthodes sont **Broker()**, constructeur par défaut qui met en relation le client pour son enregistrement auprès d'un fournisseur, cet enregistrement correspond à l'attribution d'un identifiant client par le fournisseur ; **malloc()**, méthode qui retourne au client un nouvel objet de classe Space représentant l'espace virtuel de stockage de la taille souhaitée. La classe Space ne contient que trois méthodes : **put()**, qui permet de stocker un bloc ; **get()**, qui permet de récupérer un bloc préalablement stocké ; **free()**, qui permet de libérer un bloc préalablement stocké. Les informations permettant de récupérer les blocs stockés sont enregistrées dans l'objet Space. La sauvegarde de cet objet sur disque local permet de récupérer les blocs ultérieurement.

3. Architecture

La figure 1 présente l'architecture globale de Us. Pour illustrer son utilisation, prenons un exemple de fichier distribué qui utilise Us : un utilisateur envoie son fichier au système de fichiers, lui même **client** de Us. Ce fichier est découpé en *blocs*, envoyés à un **fournisseur** de service Us. Ce dernier détient les informations sur les ressources disponibles et découpe les blocs en *fragments* en incluant des fragments de redondance générés par l'algorithme de Reed-Solomon [4]. Finalement le fournisseur répartit ces fragments sur des pairs dédiés au stockage, les **stockeurs**, qui sauvegardent localement les données. Du point de vue global, Us se compose de 3 types d'applications : le **client** qui est l'interface utilisateur pour demander de stocker des blocs, de les récupérer ou de les effacer ; le **fournisseur** qui est le fournisseur de service, il détient et gère les informations sur les ressources et les gère ; le **stockeur**, le pair qui enregistre les données sous forme de fragments. Un pair peut-être à la fois client et stockeur.

3.1. Fonctionnement

Us dispose de la classe **PeerInfo** qui contient toutes les informations permettant de communiquer avec un autre pair. Elle contient également un attribut qui indique l'état du pair : opérationnel, déconnecté ou définitivement mort. Le principe d'utilisation de cette classe est le suivant : chaque pair construit son propre PeerInfo et l'envoie aux autres pairs pour qu'ils puissent communiquer entre eux.

3.1.1. Initialisation

Chaque nouveau stockeur Us contacte le fournisseur de service qui les enregistre en sauvegardant leur PeerInfo respectif et leur attribue un identifiant. Au sein du fournisseur de service se trouve un gestionnaire de ressources qui s'occupe aussi des allocations et attributions des espaces de stockages virtuels. Chaque stockeur sauvegarde localement son identifiant pour permettre lors d'une déconnexion temporaire de pouvoir réapparaître lors de sa prochaine connexion avec le même identifiant auprès du fournisseur qui grâce au gestionnaire vérifie l'existence de cet identifiant. Si cet identifiant existe dans le gestionnaire, alors le stockeur envoie juste son nouveau PeerInfo. Par contre si cet identifiant n'existe plus, alors le stockeur sera considéré comme nouveau, et par conséquent nouvel enregistrement et réattribution d'un nouvel identifiant.

3.1.2. Ecriture

Lorsqu'un client contacte le fournisseur pour l'obtention d'un espace, le fournisseur contacte le gestionnaire qui attribue un espace qui est envoyé au client. A partir de ce moment, lorsque le client veut stocker des blocs dans cet espace nouvellement créé, il émet une requête au gestionnaire pour obtenir une liste de stockeurs. Cette liste sera envoyée avec le bloc à stocker à une entité secondaire du fournisseur, le **répartiteur** de fragments : il découpe le bloc à stocker en plus petits blocs, nommés fragments, eux-mêmes répliqués suivant l'algorithme de Reed-Solomon. Tous ces fragments sont alors dispersés sur des stockeurs référencés par le gestionnaire. Le choix de ces stockeurs est cyclique et le gestionnaire prend en compte le fait de ne pas mettre deux fragments d'un même bloc sur le même stockeur afin de maximiser la tolérance aux pannes. Si un stockeur meurt pendant le transfert d'un fragment, alors un nouveau stockeur est donné par le gestionnaire pour stocker ce fragment.

3.1.3. Lecture

Lorsqu'un client demande la récupération d'un bloc précis, il contacte un répartiteur qui récupère un nombre suffisant de fragments et reconstruit ce bloc suivant l'algorithme de Reed-Solomon pour le renvoyer au client. Dans le système de stockage Us, c'est l'algorithme de Reed-Solomon qui calcule les r fragments de redondance à partir des s fragments constituant le bloc d'origine. Lors de la reconstruction du bloc, il sera plus coûteux en temps si on ne peut pas obtenir les s fragments d'origine, car cette méthode va recalculer ces s fragments à partir de s fragments parmi les s+r fragments. Pour la lecture, la notion d'échec et liée au temps d'accès au bloc : le délai avant lequel on considère qu'il est impossible de récupérer le bloc. Le temps de lecture d'un bloc est variable, il dépend de l'action de récupération des s fragments : s'il manque un ou des fragments s, on ajoute à cette action la récupération du nombre de fragments manquants parmi les r fragments disponibles et la reconstruction des s fragments d'origine. Si cette dernière action échoue, un délai d'attente des stockeurs non connectés disposant des précieux fragments est appliqué. Passé ce délai, on considère que la lecture de ce bloc est un échec.

4. Déploiement

Us dispose de trois modes de fonctionnement : le mode autonome, le mode Internet/XtremWeb pour utiliser Us à travers Internet en passant par XtremWeb et enfin un mode Internet/Jxta. Us dispose aussi d'une interface graphique permettant de suivre l'évolution du système, comme présenté à la figure 2.

4.1. Autonome

Chaque pair doit lancer deux processus java : le **client**, interface utilisateur de Us, ainsi que le **stockeur**, processus de stockage contenant deux threads : un thread d'écriture sur le disque local et un thread de réparation. Indépendamment, le **fournisseur** doit être lancé sur un serveur.

Le mode de communication que nous avons choisi est le Java-RMI qui permet d'exécuter une méthode sur un processus distant. Il suffit de faire tourner un processus de démon et d'y inscrire le processus appelé : un thread est alors créé, rendant transparent les communications entre le démon et le processus. Cette bibliothèque intègre la gestion de l'envoi et réception des données, et le contrôle du processus distant. Ce choix a été guidé par l'extrême simplicité d'utilisation du RMI, malgré ses mauvaises performances. Le RMI est une bibliothèque bien adaptée dans son principe, mais elle manque de fonctionnalités [2]. Nous en avons conclu qu'il fallait écrire notre propre mode d'appel de méthodes à distance, avec un mode asynchrone et sans démon. Un nouveau prototype est en cours d'écriture.

4.2. XtremWeb

Pour l'utilisation de Us à travers Internet, XtremWeb [3] est mis à contribution en tant que fournisseur de ressources. XtremWeb est une plate-forme expérimentale de calcul distribué Pair à Pair. Les participants d'XtremWeb coopèrent dans un but commun, partager leur temps CPU. Cette plate-forme permet d'exécuter des programmes de manière distribuée sur un grand nombre de ressources. XtremWeb est composé de processus **client**, **serveur** et **worker**.

Pour que Us soit inter-opérable avec XtremWeb, un client XtremWeb a été créé spécifiquement. Une tâche XtremWeb est un travail paramétré associé à un binaire : dans notre cas, le binaire associé à chaque Tâche XtremWeb est le programme du stockeur Us. Par conséquent, les pairs de Us sont des workers d'XtremWeb qui ont chacun un stockeur Us comme application locale à exécuter. Le fournisseur de Us est un client d'XtremWeb qui demande des ressources, c'est-à-dire l'exécution de stockeurs Us sur les workers d'XtremWeb. Pour lancer XtremWeb et Us, il faut d'abord exécuter un serveur XtremWeb, puis un worker XtremWeb pour chaque individu voulant participer, lequel va s'enregistrer auprès du serveur XtremWeb. Ensuite il faut exécuter un Client XtremWeb, le fournisseur de service Us, qui va contacter le serveur XtremWeb pour convertir les workers XtremWeb en Stockeurs Us. On exécute un Client Us, lequel va contacter le fournisseur de service Us pour stocker des données. Chaque stockeur Us, engendré automatiquement sur un worker XtremWeb, contactera le fournisseur de service Us pour s'enregistrer et ainsi créer la liste des ressources.

4.3. Jxta

La plupart des systèmes Pair à Pair existants ont développé leurs propres protocoles pour la découverte et l'exploitation des ressources disponibles sur le réseau. Le projet open source JXTA [1], initié par Sun, propose une plate-forme générique dont l'objectif est de servir de couche de base pour la définition de services pair à pair spécialisés. JXTA propose un ensemble de protocoles de base pour la découverte et le monitoring des ressources, pour la communication inter-pairs en environnement hautement dynamique et permet à l'utilisateur de définir ses propres services. Notons que la découverte des ressources peut se faire aussi bien en mode centralisé que par diffusion.

Us utilise le protocole de découverte des pairs de Jxta pour fonctionner à travers Internet. Comme les deux précédentes versions utilisent un serveur identifié pour l'enregistrement, nous utilisons ici le mode de découverte des pairs par diffusion. En ce qui concerne les communications, un projet de portage de RMI sur Jxta existe, mais il semble qu'il n'y a toujours pas de version disponible. Nous ne pourrons donc utiliser les moyens de communications de Jxta qu'après avoir modifié le prototype avec nos propres invocations distantes basées sur de simples envois de messages : nous pourrons alors utiliser les tubes de communication de Jxta.

5. Conclusion

Us est un prototype qui permet le stockage sur un environnement pair à pair. Il est étudié pour pérenniser les données. Nous avons déjà déployé Us en mode autonome avec RMI, il existe aussi une version interfacée avec XtremWeb et une version qui utilise la découverte de ressources de Jxta. Nous n'avons pas encore étudié son déploiement sur d'autres systèmes pair à pair. Nos objectifs sont d'affiner le prototype en déployant notre propre invocation de méthodes à distance à la place de Java-RMI, en intégrant d'autres algorithmes de redondance comme Tornado et en analysant d'autres schémas de détection des pannes.

Bibliographie

1. Project jxta. http ://www.jxta.org.
2. Gil Utard Cyril Randriamaro, Olivier Soyez. Us : Prototype de stockage pair à pair. Technical report LaRIA 2003-09, Laboratoire de Recherche en Informatique d'Amiens, September 2003.
3. Gilles Fedak, Cécile Germain, Vincent Néri, and Franck Cappello. Xtremweb : A generic global computing system. In *CCGRID2001, workshop on Global Computing on Personal Devices*, May 2001.
4. James S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software Practice and Experience*, 27(9) :995–1012, 1997.