Université Pierre et Marie Curie

LiP6 - Laboratoire d'Informatique de Paris VI

Un système coopératif pour la localisation des miroirs sur le Web

THÈSE

présentée et soutenue publiquement le 20 décembre 2000
(version soutenance)

pour l'obtention du

Doctorat de l'Université Pierre et Marie Curie
(spécialité systèmes informatiques)

par

Neilze DORTA SANTIAGO PEREIRA

Composition du jury

Président :	Christian QUEINNEC
Rapporteurs :	Guy BERNARD Michel RIVEILL
Examinateurs :	Claude GIRAULT Ian PIUMARTA Pierre SENS Marc SHAPIRO

Institut National de Recherche en Informatique et en Automatique - INRIA Rocquencourt

À Edier et Nedir

Quand je parlerais la langue des hommes et des anges, si je n'ai pas l'amour, je ne suis qu'un arain qui résonne, ou une cymbale qui retentit.

1 Corinthiens 13.1

Remerciements

Je remercie cordialement Claude Girault, Professeur à l'Université Pierre et Marie Curie, qui a accepté d'être le directeur de cette thèse. Mon professeur depuis la maîtrise, c'est à lui que je dois mon intérêt pour les systèmes informatiques.

Je remercie sincèrement Guy Bernard, Professeur à l'Institut National des Telecommunications, et Michel Riveill, Professeur à l'Université de Nice, pour l'intérêt qu'ils ont porté à mon travail en acceptant de rapporter sur cette thèse.

Je tiens à remercier Christian Queinnec, Professeur à l'Université Pierre et Marie Curie, Marc Shapiro, directeur de recherche à Microsoft Research Ltd. et Ian Piumarta, docteur à l'INRIA d'avoir accepté d'être membres de mon jury.

Je tiens tout spécialement à remercier Pierre Sens, maître de conférences à l'Université Pierre et Marie Curie, pour le soutien unique qu'il m'a accordé. Je lui suis reconnaissante pour ses commentaires indispensables et pour ses critiques qui m'ont permis d'améliorer mon travail.

Je remercie Marc Shapiro, Directeur de recherche à l'INRIA, ancien responsable scientifique au sein du projet SOR, pour m'avoir accueillie dans son équipe et Mesaac Makpangou pour avoir éveillé en moi l'intérêt envers le sujet de cette thèse.

Je suis reconnaissante à l'INRIA, qui est une structure plaisante pour travailler et mener à bien ses recherches. Les ressources disponibles permettent aux chercheurs de développer leurs projets librement.

Je remercie Bertil Folliot pour ces recommandations utiles. Je remercie toute l'équipe SOR pour leur soutien et en particulier Carine Baillarguet qui a bien voulu participer à la relecture de ce manuscrit.

Je remercie tout particulièrement mes parents, Edier et Nedir, pour leur soutien inconditionnel dans l'entreprise de ce projet. Depuis toujours, ils n'ont cessé de contribuer à créer un environnement favorable à ma formation.

Je remercie chaleureusement Christian pour son soutien, sa compréhension et tous les efforts qu'il a consentis cette dernière année. Et enfin, je remercie Lucas qui dans sa première année de vie a dû partager sa maman avec cette thèse.

Résumé

La demande de partage d'information sur Internet croît avec l'utilisation de plus en plus fréquente d'applications coopératives. La répartition des utilisateurs à travers le monde et leur multiplication génèrent des problèmes de disponibilité des données et de performance, dus notamment à la latence du réseau, à la surcharge des serveurs et du réseau. La réplication des informations sur des serveurs proches des utilisateurs finaux remédie à ce problème en limitant la distance parcourue et la charge ajoutée au serveur source ou au réseau en général.

Pour être efficace, la réplication nécessite de la part de l'utilisateur la localisation transparente des copies les plus appropriées. Un système capable de découvrir des nouvelles copies de façon dynamique et de les publier est fondamental. De plus, pour assurer une localisation transparente, un système de nommage qui supporte la réplication est necéssaire. Cette thèse s'intéresse à la réplication persistante de miroirs et plus particulièrement à la publication, la découverte et la localisation transparente de ces réplicats sur le Web.

Notre solution, baptisée $AR\tilde{A}$, utilise la coopération par échange de messages entre utilisateurs pour localiser les miroirs. Elle utilise également la notion de localité d'intérêts entre utilisateurs et de proximité géographique pour limiter cette coopération à des communautés d'intérêts restreintes. La redirection locale des requêtes des utilisateurs, l'utilisation des préfixes des URLs et la gestion des aliases pour le nommage des réplicats garantissent un accès transparent aux différentes copies. Enfin, le passage à l'échelle est assuré grâce au caractère épidémique, asynchrone et paresseux du protocole de coopération, associé au principe de localité.

Une méthodologie d'évaluation spécifique a permis d'évaluer l'intérêt des miroirs et de valider le modèle proposé. D'après les mesures de performance réalisées, cette solution semble répondre favorablement au problème de localisation transparente des miroirs, et passer à l'échelle.

Mots-clés: réplication, miroirs, large échelle, localisation, modèle de coopération, protocole épidémique.

Abstract

Over the last few years, large scale information sharing has risen to dominance throughout Internet with the advent of cooperative applications. The exponentially increasing number of users scattered all over the world highlights performance and information availability problems, due to access latency, high server load and network congestion. Information replication in the vicinity of end users is a commonly used technique for improving availability and boosting performance.

For replication to be effective, users should first be able to transparently locate copies that better suit their needs, i.e nearby for better access time or the most recent version for consistency reasons. Firstly, it is fundamental to have a system capable of dynamically discovering and advertising freshly created copies. Secondly, a naming system supporting replication is needed to ensure transparency. In this thesis, we are interested in persistent replication of mirrors, and more particularly in discovering, advertising and transparently locating replicas over the Web.

Our solution, $AR\tilde{A}$, uses cooperation by exchanging mirrors location messages between users. Cooperation is limited to groups of users having common interests and who are geographically close. Transparent and local redirection of users requests, using URLs prefixes and handling aliases for naming replicas guarantees access transparency. Finally, the system scales geographically due to the epidemic, asynchronous and lazy nature of the cooperation protocol, coupled with the principle of locality.

We propose an evaluation methodology we used to show the need for mirroring and to validate to proposed system. Performance measurements show encourouging results in favour of using $AR\tilde{A}$ for transparently locating mirrors in large scale information systems.

Keywords: replication, mirrors, large scale, location, cooperation model, epidemic protocol.

Table des matières

Table des figures

Liste des tableaux

Introduction

Le Web, l'ingénierie coopérative ou les jeux sur Internet sont des applications qui présentent une demande croissante de partage d'information sur Internet. Les utilisateurs de ces applications sont souvent éloignés, ce qui introduit une latence importante pour accéder aux données partagées et diminue ainsi la qualité du service offert par ces applications. La congestion des liens et la surcharge des serveurs les plus populaires contribuent également à la dégradation des performances. Ces applications présentent donc un besoin fort de disponibilité des données couplé à de bonnes performances, besoin qui peut être caractérisé et résumé par la problèmatique d'amélioration des temps d'accès des utilisateurs aux données.

Il existe deux approches pour résoudre ce problème. Une première approche consiste à augmenter la vitesse du réseau pour diminuer le temps de latence d'un point à l'autre. Cette approche ne nous intéresse pas directement puisqu'elle ne concerne que les technologies réseaux de bas niveau (ou le déploiement de ces technologies). Une deuxième approche est de répliquer les données dans le but soit de les placer plus près de l'utilisateur soit de répartir la charge d'un serveur entre plusieurs serveurs.

Dans cette thèse nous considérons la réplication de données pour améliorer le temps d'accès des utilisateurs. Notre intérêt porte plus spécifiquement sur la réplication persistante à l'aide de miroirs et en particulier sur la localisation de ces réplicats sur le Web.

1 Réplication

Un service ou une ressource est répliqué quand il en existe plusieurs instances identiques dans différents nœuds du système. La réplication contourne le problème de la latence en plaçant les données plus près des utilisateurs diminuant ainsi leur temps d'accès. Ceci a comme effet secondaire la diminution du trafic sur certains liens menant aux serveurs et de la charge de ces derniers. La réplication permet aussi la tolérance aux fautes et l'augmentation de la performance par le coût de la duplication.

La réplication présente un certain nombre de difficultés et notamment des problèmes de cohérence entre les différentes copies, de placement de ces duplicatas sur le réseau, de découverte et de localisation de ces copies. Ces problèmes peuvent être plus ou moins difficiles à résoudre selon la ressource répliquée et selon la technique de réplication utilisée. Un utilisateur (ou une entité le représentant) désirant accéder à un document répliqué doit d'abord être capable de déterminer que le document principal est répliqué, et ensuite pouvoir trouver le réplicat le plus proche au moment de cet accès.

Quelle que soit la ressource concernée, données, processus ou messages, un réplicat est une copie d'un original qui subit des modifications, et nécessite donc des mises à jours plus ou moins périodiques calquées sur l'évolution de l'original. Pour gérer ces mises à jour, il devient nécessaire de localiser l'ensemble des réplicats. La localisation des diverses copies (pour l'accès ou la gestion) et leurs mises à jours successives constituent le plus grand coût de la réplication, et bien sûr font l'objet de nombreux compromis sur le modèle de cohérence mise en œuvre, le nombre de réplicats et leurs modalités de gestion.

Les caches et les miroirs

Les caches et les miroirs sont les deux techniques majeures de la réplication de données, et se différencient sous plusieurs aspects. Un des principaux est la granularité des documents répliqués. Un miroir est le réplicat d'une sous-arborescence de données liées entre elles, tandis qu'un cache réplique des documents indépendants les uns des autres. Un autre aspect, fortement lié au premier, concerne la mise en œuvre de politiques de gestion des documents répliqués. Par exemple, lorsqu'un nouveau document est demandé (et arrivé) un cache doit évincer un ou plusieurs documents présents afin de faire de la place pour stocker le nouveau. Le choix des documents devenus obsolètes, ou l'évolution de l'intérêt porté à tout document entrant dans le cache, doit être maintenu et géré. Ce problème ne se pose pas du tout dans le cadre des miroirs. Dans le même ordre d'idées, la durée de vie d'un document dans un cache est variable et souvent sur une courte période de temps. À l'opposé, le contenu d'un miroir et les documents qui le composent présentent une durée de vie stable. Les différences entre les caches et les miroirs demandent des politiques spécifiques pour traiter les problèmes dus à la réplication.

2 Localisation à large échelle

Pour que la réplication de données soit efficace, chaque utilisateur doit être capable d'accéder aux copies proches de sa propre localisation. Un autre élément très important est le placement de ces réplicats, dont va dépendre la localisation. Aujourd'hui, la création et la localisation des miroirs se font manuellement par l'utilisateur, ce qui, paradoxalement, entraîne une perte de temps au moment d'accéder aux données. De plus, cette « complexité d'utilisation » limite l'utilisation des miroirs. Si localiser un miroir est difficile, l'utilisateur préférera, dans ce cas, accéder aux données directement à la source, exigeant plus de bande passante et renonçant à une meilleure performance. En outre, il n'existe pas de moyen de propager de façon automatique l'information sur l'existence et la localisation d'un miroir intéressant potentiellement une communauté d'utilisateurs.

2.1 Nommage

Le premier problème de la localisation est le nommage de l'ensemble des réplicats. Actuellement, les systèmes de nommage tel que DNS [Moc87] ne supportent pas la réplication des noms d'une façon dynamique et cohérente. La liaison dynamique d'un seul nom avec plusieurs adresses ou l'identification des divers noms, liés à différentes adresses, comme étant le même document, sont des choses que l'on ne sait pas encore faire dans un environnement à large échelle. Quelques solutions statiques permettent la localisation du « meilleur » réplicat [APS98] mais ne supportent pas la découverte des ressources.

2.2 Publication et découverte

Par ailleurs, il n'existe pas de vrai support pour la publication systématique et la découverte des miroirs disponibles. La publication des miroirs utilise une technique centralisée. Ce sont les serveurs source qui se chargent de la divulgation des miroirs connus. Cette solution ne permet ni un accès transparent ni une publication exhaustive de tous les miroirs existants. De plus, localiser la copie la plus « appropriée » d'un document exige des informations additionnelles (temps d'accès vers chaque copie, date de la dernière mise à jour pour chaque miroir) et des mises à jour fréquentes, qui ne sont actuellement ni stockées, ni accessibles.

Cette recherche porte sur la localisation des miroirs sur l'Internet. Une première phase de ce travail a consisté à déterminer les serveurs qu'il serait intéressant de « mirorer », le gain qu'on pourrait obtenir si ces miroirs existaient et si l'on savait les localiser. Le plus grand problème rencontré avec cette technologie est la localisation des réplicats les plus proches. Une deuxième phase d'évaluation a donc porté sur les problèmes de localisation et les gains qui pourraient être attendus d'un système offrant la localisation.

À l'aide de cette étude, nous avons proposé un protocole de localisation de miroirs. Ce dernier a été implémenté et des mesures de performance ont été effectuées.

3 Le système $AR\tilde{A}$

Nous proposons dans cette thèse un système capable de localiser, de façon transparente, les miroirs qui intéressent un groupe d'utilisateurs. L'objectif est d'exploiter les données répliquées dans des miroirs pour diminuer le temps d'accès à un document. Pour limiter l'utilisation des ressources, le temps nécessaire à la localisation et surtout passer à l'échelle, le mécanisme mis en œuvre n'est pas global et ne permet pas de localiser l'ensemble de miroirs, mais seulement ceux qui potentiellement intéresseront les utilisateurs finaux.

La solution consiste à localiser uniquement les miroirs susceptibles d'être accédés. Ne les connaissant pas à l'avance, nous nous basons sur la localité d'intérêts entre groupes d'utilisateurs pour anticiper un accès éventuel à un miroir. Pour cela, une coopération étroite entre *organismes* permet d'échanger des informations sur la localisation des miroirs connus de chacun. Nous avons développé un protocole coopératif entre organismes qui permet l'échange épidémique de méta-données[1] et fournit un service de localisation des miroirs. Un mécanisme de gestion des aliases pour les miroirs permet la localisation des réplicats ayant différents noms. Ces services sont disponibles au travers d'un support d'accès transparent aux miroirs et d'un protocole de coopération. Enfin, notre solution offre un accès transparent par le biais d'une redirection locale de la requête du proxy, tout en assurant une cohérence incrémentale. Ce protocole et mécanismes ont été réalisés au sein du système nommé $AR\tilde{A}$.

3.1 Nommage

Le système de nommage est basé sur des préfixes d'URLs permettant des liaisons entre un nom et plusieurs adresses. Pour résoudre le problème du nommage, nous travaillons avec des tables de correspondances. La correspondance se fait entre le préfixe des URLs des serveurs source et les préfixes des URLs des serveurs miroir. $AR\tilde{A}$ identifie par comparaison de préfixe, l'entrée correspondante à l'URL de la requête. Ensuite, $AR\tilde{A}$ choisit le « meilleur » miroir parmi ceux catalogués.

Les *bases de données de miroirs* (BDM) contenant les tables de correspondances sont implantées en réparti. Chaque organisme possède sa propre BDM avec une vision locale et indépendante. La répartition de ces méta-données par organisme évite d'avoir une vision globale favorisant l'aspect extensible du système. Cela profite au non besoin de cohérence entre les BDM facilitant ainsi le passage à l'échelle.

1. Données qui permettent la localisation des miroirs.

3.2 Le modèle de coopération

La localisation de miroirs est basée sur la coopération entre groupes d'utilisateurs proches géographiquement et ayant les mêmes centres d'intérêt. La communication entre les groupes se fait au moyen d'un protocole épidémique et paresseux d'échanges de messages. La coopération peut avoir lieu entre *organismes* et groupes d'*organismes*. Nous définissons la notion d'*organisme* comme une entité formée par plusieurs sites géographiquement proches et liés par des intérêts communs.

Découverte et localisation

La découverte des miroirs se fait par échange d'informations sur leur localisation avec d'autres *organismes*. Un tel mécanisme s'avère particulièrement intéressant pour des *organismes* qui partagent des intérêts communs. La coopération entre groupes d'*organismes* se fait par épidémie grâce à des *organismes* appartenant à plusieurs groupes au même temps.

Un groupe d'*organismes* forme une entité de coopération où l'échange d'information a lieu. C'est dans le groupe d'*organismes* que la découverte des miroirs se fait. Chaque *organisme* participant coopère en partageant avec le groupe la localisation des miroirs qu'il connaît. Un groupe d'utilisateurs représente donc la granularité de base de coopération, permettant ainsi la découverte, la publication et la localisation communes des miroirs potentiellement accessibles par les intervenants du groupe. Un *organisme* $AR\tilde{A}$ actif est dénommé *cellule*. Une cellule est donc un *organisme* qui utilise le système $AR\tilde{A}$ pour localiser des miroirs et qui participe aux échanges des messages avec d'autres *organismes* pour connaître et faire connaître la localisation des miroirs.

Les *groupes d'organismes* peuvent être des *groupes ouverts* ou des *groupes fermés*. Les *groupes ouverts* sont crées par un *organisme* et acceptent des abonnements d'autres *organismes*. Ces groupes peuvent cependant restreindre l'accès selon leurs propres critères. Les *groupes fermés* visent une coopération très ciblée et donc moins chère et plus efficace. La procédure d'abonnement se fait par des accords entre les administrateurs systèmes. Dans un *groupe fermé* le nombre des membres est réduit, minimisant ainsi le nombre de messages.

Protocole épidémique et paresseux

Le système $AR\tilde{A}$ utilise une communication asynchrone, paresseuse et épidémique pour assurer l'échange de messages pour la coopération. Le protocole épidémique consiste à envoyer des messages d'un point à un autre de façon paresseuse. Le point récepteur envoie à son tour le message à d'autres points, diffusant ainsi une information par contamination. Ces messages sont envoyés par *piggybacking* (ou par *batch*) de façon asynchrone. $AR\tilde{A}$ échange les informations par épidémie pour assurer le passage à l'échelle et profiter de la topologie actuelle de l'Internet. Dans un protocole épidémique basé sur un modèle de coopération où il existe des intersections entre les groupes, certaines informations ne sont pas nécessairement intéressantes. Afin de filtrer les informations à échanger avec les autres groupes, un *degré de contamination* est établi. Le *degré de contamination* est une métrique pour savoir jusqu'à quel niveau d'indirection sur la coopération, l'information est acceptable.

3.3 Cohérence incrémentale

$AR\tilde{A}$ offre la possibilité à l'utilisateur d'avoir une cohérence de vue sur les données accédées sur des miroirs. La *cohérence incrémentale* garantit que les opérations de lecture d'un objet (d'une donnée) contiennent toutes les modifications déjà observées auparavant par cet utilisateur. La *cohérence incrémentale* permet aux utilisateurs de voir un objet mis à jour de façon incrémentale par rapport au temps assurant un « non retour sur le passé ».

Pour assurer la *cohérence incrémentale* dans la vue d'un utilisateur, $AR\tilde{A}$ offre une *session de cohérence*. Un utilisateur doit manifester explicitement son désir d'avoir une cohérence de vue incrémentale par une ouverture de session. Une *session de cohérence* assure à une suite des requêtes une *cohérence incrémentale* sur les données téléchargées. Une *session de cohérence* est une abstraction pour la séquence de lectures et d'écritures effectuées pendant l'accès d'un utilisateur à un ou plusieurs objets.

3.4 Contrôle

Pour effectuer un contrôle sur la validité des données et éviter la dégradation des performances, $AR\tilde{A}$ offre trois éléments : un « Ramasse Miettes », un mécanisme de « Validation des miroirs » et un système de « Gestion de dates ». Le *ramasse miettes* attribue à chaque message un TTL (Time To Live). En outre, une vérification sur l'existence des miroirs est effectuée régulièrement. Si un miroir n'existe plus, sa localisation est supprimée de la base de données. La validation des miroirs se fait par des envois des messages ICMP ECHO REQUEST (ceux utilisés par le *ping*). Enfin, $AR\tilde{A}$ prévoit une gestion des dates afin de maintenir une cohérence incrémentale sur l'information obtenue par les méta-données. Avant d'être stockées, les dates des méta-données sont comparées avec celles des méta-données déjà connues.

4 Méthodologie et évaluation de performance

Dans cette thèse nous avons effectué des mesures visant à comprendre le comportement des requêtes Web afin de vérifier si la création de miroirs peut améliorer le temps de réponse et diminuer le trafic réseau. En outre, nous avons effectué des mesures pour vérifier l'intérêt de l'échange d'information sur la localisation des miroirs entre *organismes*.

La méthodologie adoptée se decompose en trois étapes. La première étape consiste à collecter des traces sur le trafic réseau dans l'*organisme*. Ensuite ces traces sont analysées de façon interne et externe (par rapport à d'autres *organismes*). L'analyse interne étudie le trafic d'un *organisme* en nombre de requêtes par serveur, le nombre d'octets par requête et par serveur et donne comme résultat une liste des serveurs candidats à être *miroré*. Le temps d'accès à ces serveurs est mesuré ainsi que le nombre de routeurs traversés. Comme résultat de cette analyse, une configuration de placement des miroirs est proposée. Les traces sont alors analysées à nouveau en fonction de la configuration. Les deux étapes, *analyse* et *configuration*, se succèdent au profit d'une comparaison approfondie entre le choix des miroirs, leur placement et la diminution du trafic réseau ainsi que du temps de réponse.

Ces mesures tiennent compte du coût réseau imposé par la coopération entre les organismes. Une comparaison entre le gain obtenu avec le placement des miroirs et le coût de la coopération permet de conclure sur les bienfaits de la coopération. Dans ce sens, le calcul du gain pour différents sites ainsi qu'une étude comparative des diverses solutions de placement des miroirs selon le gain global pour l'ensemble des utilisateurs ont été fait.

Les résultats de ces mesures sont encourageants. L'application de la méthodologie d'évaluation sur des traces réelles montre un gain réel dans l'utilisation de $AR\tilde{A}$ pour la localisation des miroirs par coopération entre *organismes*.

5 Plan

Ce mémoire présente l'ensemble du travail et des résultats obtenus au cours de cette thèse. Il s'articule autour du plan suivant.

La première partie donne l'état de l'art : le chapitre 1 décrit le problème du large échelle et les conséquences sur la localisation des miroirs ; le chapitre 2 explique les stratégies existantes dans les différentes solutions pour les problèmes du large échelle et résume les solutions spécifiques au problème de la localisation de miroirs sur le Web. Cet état de l'art permet d'exposer les différentes solutions reprises lors de la conception du système $AR\tilde{A}$.

La deuxième partie décrit la spécification et la réalisation de $AR\tilde{A}$: le modèle de base utilisé par $AR\tilde{A}$ ainsi que son architecture sont présentés dans le chapitre 3 ; l'implémentation et les choix de réalisation sont décrits dans le chapitre 4.

La troisième partie étudie la méthodologie d'évaluation de $AR\tilde{A}$ et présente quelques mesures de performance : l'intérêt des mécanismes offerts par $AR\tilde{A}$ est mis en évidence dans le chapitre 5 ; le chapitre 6 rapporte les résultats et l'interprétation d'une première évaluation des performances de $AR\tilde{A}$.

La partie IV est dédiée à la conclusion de cette étude : le chapitre 7 résume les choix de conception de $AR\tilde{A}$, dégage les principales contributions de l'étude menée et propose des travaux futurs dans le cadre de ce système.

Première partie

État de l'art

Chapitre 1

Compréhension du problème

Sommaire

1.1 Introduction

La demande de partage d'informations sur Internet croît chaque jour. Les applications telles que le Web, l'ingénierie coopérative ou les jeux sur Internet ont besoin d'une meilleure performance et d'une plus grande disponibilité des données. Les utilisateurs de ces applications sont souvent placés loin l'un de l'autre, subissant une latence importante pour accéder aux données. La latence introduit des problèmes de temps d'accès et de temps de réponse, dégradant ainsi la qualité du travail. La performance souffre aussi de la surcharge sur l'Internet et sur ses serveurs le plus populaires.

Il existe deux approches pour résoudre le problème de manque de performance. Une approche consiste à augmenter la vitesse du réseau pour diminuer le temps de latence d'un point à l'autre. La deuxième approche est de répliquer les données dans le but de les placer plus près de l'utilisateur ou de répartir la charge d'un serveur entre plusieurs serveurs.

Dans cette thèse nous nous intéressons à la réplication de données pour augmenter la performance de temps d'accès et de temps de réponse des serveurs Web. Nous considérons spécialement la réplication persistante de miroirs. Dans ce cadre nous nous sommes focalisés sur la localisation de ces réplicats sur le Web.

1.2 Problèmes du large échelle

Nous pouvons décrire un réseau comme Internet [LCC+97] par trois dimensions [And87, Neu92] : la distance entre les nœuds, le nombre de nœuds et le nombre de frontières d'administration traversées.

Sur Internet nous trouvons des distances très importantes entre les nœuds. Ces nœuds peuvent être distants de plusieurs milliers de kilomètres. La dimension géographique d'un grand réseau a des répercussions sur le temps de propagation des messages et sur le taux de panne de communication.

Sur Internet un grand nombre de nœuds sont interconnectés. Le nombre d'utilisateurs ainsi que le volume des informations échangées est en rapport avec le nombre de machines connectées. Le nombre des machines se répercute sur la variation des temps de communication.

Le nombre des frontières administratives induit un problème de cohabitation, en particulier sur la politique de nommage des entités. Le manque d'un nommage universel pour chaque entité a une influence directe sur la variation du temps d'accès à une ressource.

Le problème de temps d'accès est dû principalement à la latence du réseau, aux pannes, aux fautes de communication et à la surcharge des serveurs. Les systèmes à large échelle souffrent de problèmes de performance et de disponibilité dus aux grandes distances géographiques entre les sites, au grand nombre des sites et donc leur plus grande probabilité des fautes de communication, des pannes et de déconnexion.

1.2.1 Latence

Dans le Web, le problème de performance et de disponibilité se traduit pour l'utilisateur par un accroissement du temps d'accès aux documents. Si le temps d'accès devient trop grand, le travail de l'utilisateur peut devenir difficile et improductif. En plus, l'indisponibilité d'une donnée peut occasionner des fautes et des incohérences.

Sur Internet, le temps de latence ou délai de propagation, délai nécessaire pour qu'un bit de donnée arrive à son destinataire, peut être important. Ce délai dépend de la distance parcourue pour aller d'un point à l'autre et du nombre de relais traversés. Jusqu'alors le délai de propagation était négligeable par rapport au débit de transmission [2]. Au-dessus d'un certain débit, le délai de propagation devient le facteur le plus important.

Sur Internet les nœuds peuvent être physiquement très éloignés. Lors d'une communication entre deux machines éloignées, le signal traverse une suite de supports physiques. Chacun des supports traversés a des caractéristiques différentes en terme de débit et de temps de propagation. Le temps d'un aller retour d'un message sur le Web peut ainsi être très variable. De plus, ce temps peut varier de façon significative suivant l'heure dans la journée. Cela peut être dû à une congestion ou à une panne d'un des supports traversés [BZG94].

1.2.2 Fautes

Le Web est composé par un grand nombre de machines avec un très grand nombre de liens de communication. Un système avec telles caractéristiques fonctionne toujours avec une partie de ces machines en panne. Les fautes peuvent survenir suite à une erreur de logiciel, une erreur humaine, un acte de malveillance, une surcharge du réseau ou du serveur.

Les fautes de communication peuvent rendre une ressource inaccessible et donc diminuer sa disponibilité. La disponibilité d'une ressource consiste à pouvoir l'accéder. Si une ressource est

2. Temps nécessaire au système de mettre sur le réseau un bit.

en panne ou inaccessible elle n'est plus disponible. Lorsque le nombre de ressources diminue, le temps d'accès croît de manière significatif. Pour augmenter la disponibilité d'une ressource, un réplicat de la ressource originale est crée et peut être accéder en alternative à la ressource en cas de panne. Encore faut-il qu'un chemin existe et que le réseau ne soit pas partitionné. Plus il y a des réplicats d'une ressource, plus grande est la disponibilité de cette ressource en cas de panne. Si, au contraire, la ressource est valide mais elle n'est pas accessible à cause d'un partitionnement réseau, un réplicat ne peut augmenter la disponibilité que si il est bien placé dans le réseau de façon à permettre à l'utilisateur un chemin alternatif vers le réplicat.

1.2.3 Surcharge

Le large échelle souffre des problèmes de surcharge dus au grand nombre d'utilisateurs. Un serveur qui reçoit beaucoup des requêtes peut devenir très vite indisponible dû à sa contention. Cela peut décourager l'utilisateur à revisiter le site et implique des conséquences économiques importantes si il s'agit d'un site commercial. Dans le cas d'un site scientifique le coût ne peut être mesuré en argent mais en temps perdu.

Le problème de la surcharge peut générer un problème de congestion sur le réseau dû à l'augmentation du nombre des requêtes. L'utilisateur qui n'a pas eu une réponse du serveur interrogé, va relancer une requête ou même plusieurs requêtes consécutives.

Pour prévenir et diminuer la charge d'un serveur, des données redondants sont créées. Répliquer une donnée peut répartir la charge entre les différentes copies ainsi que réduire le trafic réseau si les réplicats sont placés plus proche des utilisateurs. Afin de réduire le trafic réseau dû à la surcharge, des techniques de cache sont couramment utilisées. Les caches placent les données proche des utilisateurs évitant ainsi l'envoi de la requête sur le réseau.

Nous concluons que le problème de surcharge implique des problèmes dus à la réplication des données (sujet discuté dans la section suivante).

1.2.4 Solutions pour la latence

Afin de résoudre le problème de la latence dans les réseaux, deux solutions sont étudiées. La première consiste à augmenter la vitesse réseau elle même. La deuxième réside sur la réplication [HHB96, Lit91] des données ou leur distribution.

Réseaux à haut débit

Pour diminuer la latence on peut augmenter la vitesse du réseau. Les nouvelles technologies réseaux nous permettent d'avoir des vitesses qui atteignent jusqu'à plusieurs Giga bits par second [Par94]. Les réseaux haut débit tels que ATM [Sta95], peut favoriser l'utilisation des applications sur grands réseaux.

Quelques techniques utilisées pour augmenter la vitesse des réseaux sont la largeur de bande passante ou les nœuds intelligents. Cependant, les possibilités offertes par les réseaux à haut débit sont limités. Les réseaux à large échelle sont potentiellement limités par la vitesse de traitement d'autres niveaux, en particulier celle des systèmes d'exploitation. En outre, le haut débit est limité par le temps de latence[3], par l'inadaptation des protocoles de transport et le trafic réseau.

Les réseaux à haut débit sont contraints par des réseaux lents dans le cadre des réseaux à large échelle. C'est le problème de hétérogeneité des réseaux qui empêche les données de transiter rapidement de bout en bout. Les réseaux à haut débit permettent aux données de transiter

3. Fondamentalement limité par la vitesse de la lumière.

rapidement mais les mêmes données transiteront lentement par la suite sur un réseau lent. Il faudra plusieurs années pour adapter les infrastructures des réseaux pour le support des réseaux à haut débit.

Distribution

Un service est dit distribué quand il est fourni par plusieurs nœuds. Chaque nœud est capable d'assurer le service ou une partie du service. Chaque requête peut être traitée par un des nœuds qui offre le service ou par un sous-ensemble des nœuds.

La distribution permet à l'information, qui est gérée par un service distribué, d'être répartie à travers plusieurs serveurs. Cela est important pour plusieurs raisons. Il se peut qu'il ait trop d'informations pour un seul serveur. En outre, la distribution diminue le nombre des requêtes par serveur et permet de placer l'information plus près de la partie du réseau où elle est plus fréquemment utilisée.

Réplication

Un service ou une ressource est répliqué quand il existe plusieurs instances identiques (contrairement à la distribution) dans différents nœuds du système. La réplication permet la tolérance aux fautes et l'augmentation de la performance par le coût de la duplication. Quelle que soit la ressource répliquée, données, processus ou messages, le réplicat contient la duplication d'une donnée ou la répétition d'un travail qui exige des mises à jour. Cela induit un surcoût non négligeable. Ce coût est proportionnel au nombre des copies que doit être mise à jour. L'espace de stockage supplémentaire est aussi proportionnel au nombre total de copies.

La réplication peut contourner le problème de la latence, et ainsi compenser son coût, en plaçant les données plus proches de l'utilisateur diminuant le temps d'accès et aussi le trafic réseau.

1.3 Problèmes de la réplication

La réplication présente des problèmes de cohérence entre les différentes copies, de placement de ces duplicatas sur le réseau et, de découverte et de localisation de ces copies. Ces problèmes peuvent être plus ou moins importants selon la ressource répliquée et selon la technique de réplication utilisée. Ainsi, les différences entre les caches et les miroirs résident dans les politiques spécifiques pour traiter les problèmes dus à la réplication.

1.3.1 Cohérence

Quand un réplicat est créée, une modification de donnée faite à la source[4] peut engendrer des incohérences. Un objet répliqué peut logiquement être vu comme un seul objet. Si une modification est faite sur l'objet, la modification doit être visible dans tous les autres objets. Un ensemble des réplicats est cohérent si à un moment donné la valeur de l'objet est la même pour tous les lecteurs. Si la valeur n'est pas la même dans tous les réplicats, cet objet est dit incohérent. Or, la latence dans le large échelle pose en soit seul un problème de cohérence dû au temps nécessaire pour une mise à jour. En outre, le coût qui engendre une mise à jour constante peut être prohibitif. Cela peut justifier un choix de cohérence moins fortes.

4. Copie primaire de la donnée sur laquelle s'effectue les opérations d'écriture.

1.3.2 Placement

Le placement des réplicats doit considérer trois paramètres : (i) réduire la distance vers le plus grand nombre d'utilisateurs, (ii) augmenter la disponibilité et (iii) répartir la charge.

Le placement des réplicats dépend de l'objectif de la réplication de la ressource. Si la ressource est répliquée pour augmenter la disponibilité face à un partitionnement de réseau, alors les réplicats doivent être dispersés sur le réseau et placés de telle sorte qu'un partitionnement de réseau ne rendra pas les données inaccessibles pour la majorité des utilisateurs. Si les utilisateurs sont dans leur majorité placés sur des sites proches, alors la création des réplicats vise une plus grande fiabilité ou une plus grande disponibilité en cas de défaillance d'un serveur local ou une répartition de la charge entre plusieurs serveurs. Dans ce cas, les réplicats doivent être placés prés les uns des autres. Si la ressource est répliquée visant une diminution du temps d'accès, alors les réplicats doivent être placés le plus près de la majorité des utilisateurs.

Le placement des réplicats est décisif pour le choix de la politique de cohérence et de la politique du choix du réplicat à accéder.

1.3.3 Découverte

La découverte des réplicats consiste à trouver l'adresse de chaque copie d'une ressource. La découverte des réplicats est nécessaire si le système de nommage ne tient pas compte des similitudes entre les ressources. Dans ce cas, un système capable de connaître et faire connaître l'adresse de ces réplicats est nécessaire afin de lier explicitement les ressources similaires. Le lien entre les ressources permet à l'utilisateur d'accéder à une de ces ressources au choix.

Nommage

Actuellement, les systèmes de nommage tel que DNS [Moc87] ne supportent pas la réplication des noms d'une façon dynamique et cohérente. La liaison dynamique d'un seul nom avec plusieurs adresses ou l'identification des divers noms, associés au même document reste un problème ouvert dans un environnement à large échelle. Quelques solutions statiques permettent la localisation du 'meilleur' réplicat [APS98] mais ne supportent pas la découverte des ressources.

Par exemple, il n'existe pas de vrai support pour la publication systématique et la découverte des miroirs de serveurs Web disponibles. Ce sont les serveurs source qui se chargent de la divulgation de leurs miroirs quand ils les connaissent. Cette solution ne garantit pas la connaissance de tous les miroirs existants accessibles à un groupe d'utilisateurs.

1.3.4 Localisation

La localisation d'un réplicat est nécessaire à chaque fois que la copie à accéder change. L'utilisateur doit choisir, parmi les copies qu'il connaît, la copie qu'il va accéder. Le service de localisation peut contenir un service de localisation transparent intelligent dans le choix du ressource à accéder. Le service de localisation transparent offre l'accès à ces ressources [GL91] comme étant un seul. L'utilisateur accède aux ressources indifféremment sans savoir lequel d'entre eux est accédé. Le choix du ressource à accéder se base sur des critères d'optimisation du temps d'accès ainsi que sur la cohérence entre les diverses versions de la donnée.

Un service transparent est important car il permet un gain de temps considérable à l'utilisateur. Cela implique, bien sûr, que le service d'accès transparent soit conçu de sorte à savoir choisir la copie la plus appropriée.

Sans le service d'accès transparent, il devient difficile de tirer profit de la réplication. L'utilisateur préférera accéder la donnée à l'adresse déjà connu sans tenir compte de la proximité ou de la charge. Cela implique donc que tout service d'accès doit connaître l'existence et la localisation des réplicats en s'appuyant sur un service de découverte et de localisation des réplicats.

1.4 Problème de découverte et localisation des miroirs

Dans le cadre de cette thèse, nous nous intéressons particulièrement à la localisation des différentes copies d'une donnée, les difficultés pour découvrir l'existence d'un nouveau réplicat ou simplement pour localiser le réplicat le plus approprié. Cette thèse ne traite pas de la cohérence des objets mais juste de leur localisation et placement. De plus, les solutions proposées se focalisent sur les problèmes de localisation des miroirs et n'adressent pas les problèmes des caches.

La localisation des miroirs sur Internet est un problème difficile : (i) chaque miroir est désigné par un nom différent ; (ii) aucun système de publication n'existe pour faire connaître un nouveau miroir ; (iii) il n'y a pas de mécanisme qui permet la redirection d'une requête vers le miroir le plus « approprié » de façon transparente.

Plusieurs études ont été faites concernant les caches [BO99, TDVK98, Pie99] et très peu d'études sur les miroirs [KRR99, Aka, BB99]. Les miroirs, contrairement aux caches, est la réplication d'une sous-arborescence entière des documents. Quand on a un miroir, les documents ont des noms différents par rapport à ceux de la source. Quand on a un ensemble des miroirs d'une même source, les documents ont des noms différents selon le miroir. Cela est dû au fait que la localisation des données fait partie des noms des documents sur Internet. Localiser un miroir est difficile parce qu'il n'existe pas un service capable d'identifier de façon unique les miroirs correspondants à une même source. Un autre problème non résolu est de découvrir les nouveaux miroirs et de détecter la disparition de ceux déjà existants.

Découverte des miroirs

Actuellement, aucun système permet de découvrir les miroirs dynamiquement et d'une façon décentralisée. La découverte d'un miroir est gérée par les serveurs sources eux-mêmes. Les serveurs sources publient dans leur page Web la localisation de leurs miroirs. L'utilisateur peut ainsi connaître l'existence des miroirs d'un serveur et choisir de les accéder à leur place.

Cette méthode présente des problèmes de performance et de transparence. De plus, cette solution ne tient pas compte des miroirs « marginaux » : les miroirs crées par d'autres que ceux responsables du serveur source.

Localisation des miroirs

Actuellement, la localisation des miroirs se fait manuellement par l'utilisateur ou à l'aide du serveur source lui-même que publie les adresses de ses miroirs connus. Dans la figure 1.1 un utilisateur émet une requête au *serveur source A*. Le *serveur source A* répond avec sa page principale contenant les adresses de tous ses miroirs. L'utilisateur choisit par la suite le miroir le plus proche de son site et lui envoie une requête. Le *miroir A1* répond avec les données demandées.

Cette technique entraîne une perte de temps au moment d'accéder la donnée. On peut aussi citer le manque de motivation pour accéder un miroir de la part de l'utilisateur. Si localiser un

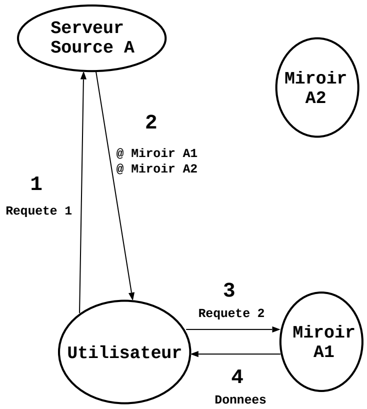

FIG. 1.1 – *Localisation de miroirs*

miroir peut être difficile, l'utilisateur préférera, dans ce cas, accéder les données directement à la source, exigeant plus de bande passante et sacrifiant à une meilleur performance.

La localisation du réplicat le plus proche de l'utilisateur final est un des éléments essentiels de la réplication car elle assure l'efficacité du mécanisme en reliant l'utilisateur à la copie la plus proche (et éventuellement la plus disponible) ou la copie ayant une version récente de la donnée. En plus, il est souhaitable de localiser les réplicats de façon transparente.

La localisation est une couche du système qui a comme fonction de trouver les objets stockés en mémoire ou sur disque. Ce service est basé sur des références des objets, soit une adresse mémoire, une adresse disque ou, dans notre cas, une adresse Internet liée à un nom qui identifie l'objet lui même.

Comme le montre le schéma ci-dessous, le service de localisation est placé en dessous du service de réplication. Que la donnée soit ou non répliquée, il faut la localiser. Sans le service de localisation, aucun donnée n'est utilisable. Dans notre cas, les services de localisation existants ne suffisent pas. Ces services ne traitent pas les données présentant plusieurs copies et sont donc incapables de reconnaître deux objets identiques avec des adresses différentes.

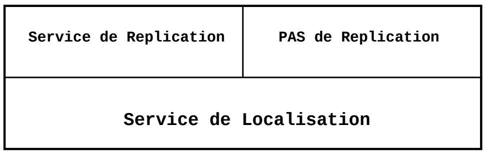

FIG. 1.2 – *Service de localisation*

La localisation des réplicats demande un système qui puisse découvrir des nouvelles copies de la donnée de façon dynamique. Dans cette thèse, nous nous sommes intéressés à localiser les miroirs de façon transparente à travers l'Internet. Mon travail consiste en fournir un accès transparent aux miroirs. Ce travail se base sur la redirection des requêtes http [BL95].

1.5 Résumé

L'accroissement de la demande de partage de l'information sur Internet, liée aux grandes distances géographiques et au grand nombre des machines, créent des problèmes de surcharge du réseau et des serveurs, de temps d'accès et de pannes.

Pour faire face à ces problèmes, nous utilisons la réplication de données dans le but de diminuer la distance entre l'utilisateur et les données , partager la charge entre les différentes copies, résister au partitionnement réseau et offrir une tolérance aux fautes.

Les techniques de réplication posent des problèmes de cohérence entre les différentes copies, de découverte et localisation de ces copies et aussi de placement de ces copies sur le réseau.

Il existe deux techniques majeures de réplication : les caches et les miroirs. Le cache se situe entre les serveurs et les clients. Il intercepte les requêtes des clients, les transmet aux serveurs, en reçoit les réponses des serveurs et les transmet aux clients. Les caches stockent (sur disque et/ou en mémoire) localement les couples réponse/requête. Cela permet de réduire le nombre de requêtes à destination d'un serveur. Le principe de fonctionnement des caches repose sur les propriétés de localité du trafic.

Le miroir est un réplicat d'une sous-arborescence de données liées entre elles. Généralement, il s'agit de la réplication des serveurs Web sur des sites proches de leurs utilisateurs. On économise ainsi en temps d'accès si le miroir est placé sur un site plus proche du client que le serveur source et si, bien sûr, on connaît la localisation du miroir.

La découverte des miroirs est un problème non résolu. Ce sont les serveurs source qui publient sur leurs sites la localisation de leurs miroirs. L'utilisateur doit alors localiser les miroirs manuellement en interrogeant d'abord le serveur source. Le coût de la localisation peut devenir prohibitif.

La localisation des miroirs a donc besoin d'un système dynamique de découverte et localisation des nouvelles copies, système que nous proposons dans la deuxième partie.

Chapitre 2

Stratégies de solutions existantes

Sommaire

2.1 Introduction

Les problèmes discutés dans le chapitre 1 sont présents dans tous les systèmes qui utilisent la réplication comme solution pour réduire la latence et la surcharge. Les systèmes de fichiers répartis, les systèmes d'exploitation répartis ou les systèmes de base de données répliquées, ont adoptés des techniques capables de résoudre les problèmes dus à la réplication et au large échelle.

Dans ce chapitre, nous présentons les stratégies existantes pour gérer les problèmes liés au large échelle. Pour chaque solution, nous donnons des exemples des systèmes qui ont choisi cette stratégie dans sa conception et nous expliquons leur fonctionnement. Cela permettra au lecteur de mieux comprendre par la suite les choix de conception de $AR\tilde{A}$. Ensuite, nous expliquons les différentes entités d'un système de désignation et nous présentons les principaux systèmes de localisation et de découverte dans le large échelle. La dernière section est dédiée à la présentation des différents systèmes de localisation des miroirs en particulier. Pour finir nous comparons ces systèmes aux solutions choisies dans la conception de $AR\tilde{A}$ et nous concluons.

2.2 Solutions pour le large échelle

Pour faire face aux problèmes du large échelle, des solutions capables de s'adapter au grand nombre des sites ainsi qu'aux grandes distances géographiques est nécessaire. Au niveau architectural, deux approches sont intéressantes : la disposition hiérarchique et le modèle de coopération par groupe. Pour contourner le problème de latence deux techniques sont efficaces : la communication asynchrone et la réplication de données. Cette dernière pouvant faire face aussi aux problèmes de pannes de machine et de partitionnement de réseau.

2.2.1 Modèle hiérarchique

Le modèle hiérarchique vise à distribuer les communications et les responsabilités administratives. Organiser les sites de façon hiérarchique diminue le nombre des requêtes adressées aux niveaux supérieurs de l'hiérarchie. En outre, avec plusieurs niveaux, on canalise les communications, diminuant le nombre des sites à connaître dans l'ensemble total des sites. Au lieu de connaître tous les sites existants, chaque site connaît ses ascendants et ses descendants hiérarchiques directes. Pour envoyer un message à tous les sites, l'émetteur du message l'envoie uniquement à ses descendants et ascendants directs. Ce mécanisme suggère une distribution des responsabilités administratives selon le niveau hiérarchique et par rapport à ses descendants. Cela décentralise la charge et permet de gérer un plus grand nombre de sites à la fois. Cependant, le modèle hiérarchique fige d'une certaine façon les chemins de communication limitant le meilleur choix d'une communication performante.

Les systèmes Globe [SHT97] et DNS [Moc87] utilisent un arbre hiérarchique pour fournir des services à large échelle. Un arbre hiérarchique suppose une racine où l'ensemble des informations sont centralisées. DNS propose un espace de nommage organisé en forme d'arbre hiérarchique et divisé en domaines (voir la section 2.4). Globe propose un service de localisation pour le large échelle par le biais d'un arbre universel de recherche distribué en régions (voir la section 2.4).

Harvest [CDN+95] est un système de caches coopératifs hiérarchiques qui permet la localisation des documents de façon plus efficace sur le Web. C'est l'architecture en hiérarchie des caches coopérants qui permet à Harvest de passer à l'échelle.

2.2.2 Modèle de coopération

La coopération est une technique qui permet à des groupes d'entités de partager des informations. Le modèle de coopération est basé sur l'idée simple de la décentralisation. Le modèle de coopération se base sur l'échange des informations par des entités qui présentent des intérêts communs et qui sont proches géographiquement. La coopération est un modèle extensible. L'extensibilité est la capacité d'intégrer de nouveaux participants au modèle sans pour autant être obligé de le reformuler ou de le redimensionner. Cette facilité de gestion favorise l'utilisation de ce modèle dans le large échelle.

La coopération consiste à partager des informations dans un groupe, évitant ainsi que chacun de ses membres cherche l'information à l'extérieur de celui-ci. On réduit ainsi le nombre de messages en circulation entre les groupes. Cela permet de réduire la bande passante utilisée dans le réseau et la charge des requêtes sur les serveurs. De plus, la coopération permet un gain en temps de réponse par les membres du groupe qui bénéficient des informations échangées par un autre participant du groupe. La coopération peut se faire par une *communication de groupe* ou par des *communications point à point*.

Relais [MPKD98, MrB97] utilise la coopération entre les caches pour localiser plus rapidement et à un coût plus bas les documents sur le Web. La coopération est faite à travers une communication de groupe asynchrone.

Pharos [BDM98] est un système de filtrage d'informations collaboratif. Ce système permet à des personnes, travaillant sur une même thématique, de partager leurs connaissances des ressources disponibles sur le Web. Pour cela une infrastructure d'évaluation collaborative des ressources doit être disponible. Les critères d'évaluation pouvant varier d'un domaine à l'autre, les évaluations sont réparties dans des canaux thématiques (dans le même esprit que les *news groups*). À partir des évaluations disponibles dans un canal, le système est en mesure de faire des recommandations adaptées pour chaque membre du canal. Ces recommandations peuvent être calculées selon différents algorithmes.

Le travail présenté par [Tac97, TB97, TB96] propose une architecture d'ensemble des réseaux coopérants pour localiser des objets dans un réseau étendu. Le réseau étendu est découpé en petits réseaux (réseaux locaux). La coopération se fait parmi un sous-ensemble de ses petits réseaux qui forment un graphe de réseaux coopérants.

2.2.3 Communication asynchrone

La communication asynchrone permet des échanges de messages entre entités de façon non-bloquante. Contrairement à la communication synchrone qui est bloquée ou déclenchée au moment d'un événement précis (en détriment d'autres tâches), la communication asynchrone attend le moment propice pour envoyer le message. Le bon moment peut être lorsque le serveur a moins de charge ou lorsque le serveur a un nombre significatif de messages à envoyer. Dans la communication synchrone, l'avantage est que la mise à jour de l'information est la plus rapide et l'inconvénient contre est que le système est bloquant.

L'asynchronisme des systèmes non-bloquants, permet aux applications de fonctionner correctement même dans un environnement avec un grand nombre des machines et avec des grandes distances. De plus, l'agrégation des messages économise la bande passante et optimise le temps de travail du serveur. Cependant, le décalage de temps dû à la communication asynchrone peut causer des problèmes de cohérence.

La communication asynchrone peut être une communication de groupe ou une communication point à point. Ces formes de communication peuvent servir de support pour un échange épidémique des messages.

Communication de groupe

La communication de groupe est basée sur une couche basse dans le protocole de transport. Cette couche connaît les membres du groupe et est capable d'acheminer les messages à tous les membres du groupe à un coût réduit.

La technique de la communication de groupe, dans l'implémentation centralisée, n'est pas adaptée à la tolérance aux fautes et aux partitionnements réseaux. La centralisation d'envoi des messages est un point favorable aux défaillances. De plus, si un partitionnement réseau survient il a peu de chances (sauf si un protocole est prévu pour supporter cela) que les sites ayant été séparés reçoivent le message. Dans le système Isis [Isi91, BJS87, Bir85] un protocole intègre une gestion de groupes tolérant aux fautes.

Communication point à point

La communication point à point permet à n'importe quelle machine de communiquer avec une autre machine d'une façon indépendante. Elle assure l'échange de données entre deux machines capables de communiquer. La communication point à point présente l'avantage de supporter mieux les problèmes de partitionnement réseau ou de panne de machine, par son aspect décentralisé d'envoi de messages et la communication via des machines non communicantes. Une information peut donc être transmise même si certains sites avec qui il y a coopération ne répondent pas.

Cette forme de communication permet un gain de temps considérable dans l'échange d'information. D'un autre coté, la communication point à point peut créer des incohérences entre les données de différents sites. Cela parce qu'une communication point à point englobant des machines non communicantes est forcement asynchrone.

Le système de base de données Bayou [DGH+87, PST+97] utilise la communication point à point pour assurer les échanges d'informations pour le maintient de la cohérence entre les divers réplicats de la base de données.

Échange épidémique des messages

L'échange épidémique [RGK95] assure la propagation des données entre des machines qui ne communiquent pas directement. La communication épidémique consiste à envoyer des messages d'un point à un autre point. Le point récepteur envoie lui aussi le message à d'autres points, diffusant ainsi une information par contamination « *voisin-voisin* ». C'est la communication point à point qui permet l'échange des messages de façon indirecte (par épidémie). Cette façon d'échanger des messages permet le passage à l'échelle puisqu'elle suppose un graphe extensible et mutable de communication. Elle réduit les informations concernant les destinataires, simplifiant la gestion d'envoi des messages à un coût plus bas. En outre, elle supporte l'indépendance de la communication par rapport au nombre total de sites, favorisant ainsi une vision locale. De plus, l'échange épidémique réduit le problème de partitionnement réseau dans les échanges des messages grâce aux différents chemins empruntés par les messages.

Le système Bayou [PST+97] utilise un échange épidémique dans la propagation des mises à jours des données pour assurer une cohérence faible entre ses réplicats.

2.2.4 Réplication

La réplication permet de répondre à différents types de problèmes tels que la tolérance aux défaillances (en installant des objets redondants), la latence (en installant les objets plus proches de leurs clients), la concurrence des accès (en partageant la charge entre plusieurs serveurs identiques) et le trafic réseau.

Les réplicats effectuent un travail répétitif, comme exécuter les mises à jour d'une donnée. Cela crée un coût proportionnel au nombre des copies qui doivent être mises à jour. L'espace de stockage supplémentaire est aussi proportionnel au nombre des copies.

La réplication de données placées plus proches de l'utilisateur final permet de diminuer le temps de latence et le trafic réseau. La réplication aussi augmente la disponibilité et permet une tolérance aux fautes, en redirigeant l'utilisateur vers un autre réplicat en cas de panne. La réplication est pour cela un outil incontournable dans la conception des systèmes sur le large échelle.

2.3 Réplication à large échelle

Sur Internet, il existe deux techniques majeures de réplication de données : les caches et les miroirs. Ces deux techniques sont différentes dans plusieurs aspects. Les caches sont plus proches des utilisateurs, plus rapides et moins stables. Ils sont utilisés dans des nombreuses applications qui ont une localité temporelle significative. Les miroirs sont moins nombreux mais mieux partagés sur le réseau. Ils possèdent des données plus stables et des vitesses d'accès plus performante que celle de la source, et sont la solution pour des ressources trop accédées par des utilisateurs géographiquement dispersés.

2.3.1 Caches

Le cache se place entre les serveurs et les clients. Il intercepte les requêtes des clients et les transmet aux serveurs et reçoit les réponses des serveurs et les transmet aux clients. Les caches stockent (sur disque et/ou en mémoire) localement les couples réponse/requête. On économise une requête à destination d'un serveur si une même requête a déjà été faite ultérieurement. Le principe de fonctionnement des caches repose sur les propriétés de localité du trafic.

Les caches sont une forme temporaire de réplication. Selon le protocole, les données n'ont pas besoin d'être mises à jour. À la place, les données sont invalidées quand la cohérence ne peut pas être assurée. Les caches répliquent des documents indépendants les uns des autres. Les caches évincent les documents selon certaines politiques qui ont pour but optimiser l'espace de stockage, la disponibilité et maximiser la cohérence. C'est pour cela que les données à l'intérieur d'un cache sont variables dans une courte période de temps.

La localisation sur Internet (globale) de tous les réplicats dans des caches a un coût prohibitif étant donnée que les documents changent trop souvent et que le grain de réplication est trop petit. Par contre, la localisation directement dans le cache (locale) est facilité par le fait que les documents gardent leur nom d'origine indépendamment de leur localisation dans le réseau. De plus, les caches assurent un accès transparent à l'utilisateur qui ignore si le document est caché ou téléchargé.

Le grand intérêt des caches est la proximité de la donnée de l'utilisateur final. Cependant, le placement des caches ne tiennent pas toujours compte de la topologie du réseau et de ses performances.

2.3.2 Miroirs

Le miroir est un système de substitution d'un serveur de données par un autre serveur de données plus proche de ses clients ou moins chargés. Le miroir est la forme la plus simple de la réplication. Il contient les mêmes données que son serveur source et est capable de répondre aux requêtes des clients à sa place. Il reçoit les requêtes d'un client connaissant sa localisation et lui répond. On économisera en temps de réponse plus le miroir est proche et moins chargé. L'efficacité des miroirs repose sur le nombre des requêtes émises vers le serveur source (charge du serveur), sur le volume de données (charge du serveur) demandées au serveur source et la proximité des clients (placement des miroirs) par rapport à la nouvelle copie du serveur (miroir).

Le miroir est la duplication des données géré par un serveur. La réplication des serveurs Web sur des sites proches de leurs utilisateurs peut permettre d'augmenter la disponibilité ainsi que les performances d'accès. Les miroirs peuvent aussi réduire la charge des serveurs ainsi que le nombre des messages sur le réseau. Le miroir est placé sur le réseau de sorte à réduire la distance entre les clients et les données consultées. De ce fait, les miroirs d'un serveur devront être distribués sur le réseau dans des emplacements où le nombre des clients est important et où le partitionnement du réseau peut être évité au maximum. Les miroirs peuvent aussi, dans un but de répartition de charge et de tolérance aux fautes, être placés à côté du serveur source.

Les miroirs présentent des problèmes typiques de la réplication. Les miroirs ont un maintien de la cohérence coûteux dû au gros grain des données, une localisation manuelle et centralisée qui coûte chère en performance, un placement anarchique et pas de transparence pour la localisation.

Un miroir est le réplicat d'une sous-arborescence de données liées entre elles. Contrairement aux caches, un miroir n'évince pas ses documents, à la place il met à jour leurs versions. La durée de vie d'un miroir est stable ainsi que son contenu. En générale, la cohérence entre le miroir et sa source a un coût très élevé. Étant donnée l'aspect persistant des miroirs, cela implique une mise à jour périodique des données du miroir par rapport à sa source.

Actuellement, les miroirs ne prennent pas vraiment en compte la topologie du réseau pour éviter le partitionnement et donc le manque de disponibilité de la donnée. Le placement tient rarement compte de la charge du serveur ou des débits des liaisons physiques. Les miroirs gérées par les serveurs source sont quand même distribués géographiquement dans un souci de répartition de charge entre les différents miroirs, ainsi que de rapprochement de l'utilisateur.

Enfin, la localisation des miroirs est difficile et pas transparente parce qu'elle est centralisée par les serveurs sources et exigent une participation active de l'utilisateur. En outre, la localisation des miroirs marginaux (non connus du serveur source) n'est pas possible.

2.3.3 Paramètres pour la réplication

La duplication de données présente certains paramètres qui modifient ses caractéristiques. Ces paramètres sont : la taille de la donnée à répliquer, la durée de vie de ses données, la distance de ses données par rapport à leur utilisateur final, le placement dans le graphe réseau. Dans cette section nous détaillons chacun de ses paramètres et nous citons des exemples des systèmes qui utilisent certains choix dans leur conception.

Granularité

Le grain utilisé au moment de la réplication influe dans le coût de la mise à jour de la donnée et dans le temps d'accès à la donnée. L'unité de réplication peut être un document (Web), un fichier (UNIX), un bloc de fichier, une arborescence des fichiers, un serveur, une machine ou plus généralement un objet.

- **Document**

  Plusieurs systèmes de cache utilisent le document comme unité de réplication. Nous entendons par document, un document dans le sens Web [5]. Les systèmes de cache Web travaillent sur une division logique de la donnée et aussi en respectant le système de nommage URL [URL, BLMM94]. Cela facilite un possible pré-chargement des documents appartenant au même niveau de nommage.
  Le système de coopération de cache Relais [MPKD98] est un système de localisation de documents sur le Web. *Relais* garde un répertoire contenant la localisation de la totalité des documents se trouvant dans les caches voisins. Ainsi, si le document demandé n'est pas dans le cache local, ce dernier regarde dans son répertoire pour voir si ce document se trouve dans un cache voisin ou pas. Si oui, il demande le document au cache voisin et le retourne au client, sinon il fait la requête directement au serveur de ce document.
  Harvest [CDN+95] est un système de caches coopératifs hiérarchiques qui utilise le document comme unité de localisation. Dans le cas d'une requête, si le document demandé se trouve dans le cache local, il est servi. Sinon, la requête est diffusée aux caches voisins et parents. Le premier à répondre positivement est choisi pour servir la requête, sinon la requête est redirigée vers le serveur d'origine. Cette méthode de localisation permet de savoir d'une manière assez sûre si un document existe ou pas dans le cache.

- **Fichier**

  La réplication de fichiers se justifie par la division logique de la donnée. Cette granularité est adaptée aux petits fichiers et ceux de taille moyenne qui ne sont pas long à charger. Ces types de fichiers sont aussi intéressants parce que suite à des modifications dans différentes copies, la résolution des conflits porte sur un volume faible de données. Contrairement aux petits fichiers, les grands fichiers sont long à télécharger et peuvent générer des conflits coûteux à résoudre.
  Le système AFS [How88, Sat90] est un système de fichiers réparti basé sur le modèle UNIX. Le système AFS utilise le fichier comme unité de données à garder dans le cache. Le partage d'information est supporté de façon à avoir un seul espace de noms comme un sous-répertoire du système de fichiers local. Chaque serveur contient un système de fichiers local "mirorant" la structure du système de fichiers partagé. La recherche d'un fichier ne se trouvant pas sur un serveur finirait par aboutir sur un répertoire « stub » identifiant le serveur contenant le fichier. Les clients gardent un cache de localisation par préfixe ; ils peuvent mettre en cache un fichier entier.

- **Blocs**

  Cette stratégie améliore la latence d'ouverture de fichiers et permet l'accès à des fichiers très larges par des stations de travail qui n'ont pas assez de place en cache pour mettre des fichiers entiers.
  Le système AFS-3 [How88] est un système de fichiers réparti qui utilise des blocs de fichier comme unité de données à garder dans le cache. AFS-3 comporte un changement au niveau de la gestion de la mémoire cache des clients par rapport au système AFS. Le client peut mettre en cache des blocs de fichiers (64 Koctets) alors qu'il ne pouvait que mettre en cache

5. Une page Web.

la totalité d'un fichier dans les versions précédentes.

– **Arborescence de fichiers**

La réplication d'une arborescence est intéressante parce qu'elle permet de rassembler plusieurs données liées logiquement dans une seule copie. Un utilisateur peut atteindre plusieurs données successivement en accédant à la même adresse. Les miroirs sont un exemple de réplication d'une arborescence des fichiers [6].

– **Serveurs**

La réplication d'un serveur est souhaitable lorsque ce serveur est responsable de répondre à des requêtes assez fréquentes et en grand nombre, ou encore quand il est important de diminuer le temps de latence en plaçant une copie du serveur plus proche des utilisateurs. Ce sont donc les serveurs trop chargés et trop éloignés qui sont de bons candidats à la réplication. Dans le Web, nous trouvons des serveurs répliqués (serveurs Web) partout dans le monde. Nous appelons serveurs miroirs ces copies de serveurs Web.

– **Machine**

Le grain machine est utilisé pour le nommage dans les réseaux. On utilise la machine comme unité de base dans l'adressage et dans les services de nommage dans tous les réseaux. Dans l'Internet, le système de nommage DNS (*Domain Name Server*) [Moc87] assure que chaque machine a un nom unique dans le réseau. En outre DNS fournit la correspondance entre un nom de machine et son adresse IP. Dans les systèmes à large échelle il est nécessaire de savoir localiser une machine donnée par son adresse IP et aussi par son nom. Le lien entre l'adresse IP et le nom de la machine doit être crée et stocké de façon à pouvoir répondre des requêtes sur leur nécessité de réplication.

– **Objet**

L'intérêt de travailler avec des objets est de pouvoir dissocier le fonctionnement de chaque service de l'objet lui-même. Toute la partie des protocoles de communication, stratégies de réplication, distribution et migration de l'état de l'objet font partie de l'objet lui-même et sont cachés par son interface.
L'architecture de Globe [SHT97] repose sur un modèle à objet et un support à des services de base. Le modèle à objets vise le partage d'objets dans le large échelle. Ces objets peuvent être partagé par un grand nombre des processus. Les services de base supportent entre autres le nommage et la localisation d'objet ; toujours dans le cadre large échelle.
Un objet Globe est distribué physiquement et l'état de cet objet peut être partitionné et répliqué dans plusieurs machines au même temps. Pour faire face aux problèmes dus au large échelle, le modèle à objets de Globe est basé sur les *objets distribués partagés* [MGLNS94]. Un *objet distribué partagé* est composé par des objets locaux situés dans des espaces d'adressages différents. Un objet local cache à l'utilisateur la réplication, la distribution et la migration de l'*objet distribué partagé*.

6. Des sites *ftp* par exemple.

Durée de vie

La durée de vie d'un réplicat modifie les politiques de traitement de la cohérence entre les diverses copies, les politiques de placement des copies ainsi que les mécanismes de localisation. La durée de vie d'un réplicat varie selon le type de réplicat. Ainsi, les caches présentent une durée de vie assez courte, tandis que les miroirs présentent un aspect persistant dans le temps.

Les réplicats qui ont une durée de vie courte seront, d'une manière générale, placés près des clients. Ce sont des copies créées pour une utilisation dans un futur proche et qui n'ont pas le temps de se faire connaître par des clients lointains. C'est pourquoi les mécanismes de localisation sont conçus pour traiter des informations qui changent constamment et rapidement. De même, le maintient de la cohérence entre les réplicats est adapté à leur courte durée d'existence ; soit une cohérence faible soit l'invalidation de la copie.

Les réplicats qui ont une durée de vie longue peuvent bénéficier d'un mécanisme de localisation plus élaboré. Cela permet à des clients placés à des endroits différents de connaître et d'utiliser la même copie. Dans ce sens, le placement des réplicats ayant une longue durée de vie peut se faire non localement par rapport aux clients. Le maintien de la cohérence est dans ce cas obligatoire si nous voulons avoir un réplicat à jour. Les mécanismes de maintien de cohérence faible ou de cohérence forte sont envisageables.

Politique d'évincement. Les caches Web hébergent des réplicats de courte durée de vie. Un document est caché au moment de la réponse à une requête client. Ce document restera dans le cache pendant une période qui varie selon sa taille, la fréquence d'accès à ce document et l'espace de stockage disponible. Une fonction de ces critères détermine le document à évincer du cache ; celui-ci sera remplacé aussitôt par un nouveau document.

TTL : *Time to Live.* Les caches utilisent aussi un TTL pour mieux gérer l'espace de stockage dans le cache et la cohérence de ses objets. Pour chaque objet caché un temps de vie est attribué à la fin duquel l'objet est exclu du cache. Cette politique est facile à mettre en œuvre et peut assurer une cohérence satisfaisante si le TTL est bien choisi.

Persistance. Les miroirs sont des réplicats contenant des objets persistants pendant une durée de vie longue. Souvent, un miroir présente une durée d'existence égale (ou dans le même ordre de grandeur) à celle du serveur source miroré. La persistance est surtout due au fait que les données répliquées par un miroir sont de nature peu changeantes dans ce qui concerne les entités qui le compose. Dans le cas d'une durée de vie persistante, le réplicat peut aussi avoir une localisation stable et bénéficier d'un protocole de mise à jour de son contenu.

Placement

Le placement du réplicat est fondamental pour répondre au mieux aux motivations initiales de sa création. Si la réplication a pour objectif d'assurer la tolérance aux fautes, il est intéressant de placer ses copies de sorte à éviter d'être inaccessibles pendant un partitionnement réseau et d'assurer ainsi la plus grande disponibilité possible. Disposer les réplicats de cette façon est loin d'être facile parce que cela exige une connaissance de la topologie du réseau par rapport aux clients potentiels. Si la réplication a pour but de répartir la charge, il est important de placer les copies proches de la plupart des clients. Dans le cas où la réplication vise à diminuer le temps de latence et améliorer la performance, le placement le plus proche du client est le meilleur.

Anarchique. Actuellement, les miroirs crées par des utilisateurs sont placés de façon a-narchique. Chaque université ou organisme créateur d'un miroir le place localement, visant en premier lieu les clients locaux et non pas les clients de la région. Cela ne favorise guère les critères pour accroître la disponibilité et pour répartir la charge.

Stratégique. Les caches Web sont par nature placés d'une manière stratégique. Proches des clients, les caches remplissent bien leur rôle d'améliorer la performance, diminuant au mieux le temps de latence réseau. Un autre exemple est les miroirs créées par les serveurs sources eux-mêmes et qui sont placés de façon planifiée. Ces miroirs sont normalement placés visant une distribution géographique des copies. Ces serveurs source sont en général des serveurs très populaires qui proposent des miroirs par continent voire par pays. Ces placements sont en partie réussi : d'une côté la distance entre les clients diminue et la charge du serveur source décroît ; d'une autre côté la topologie du réseau est prise en compte.

2.3.4 Cohérence

Le manque de cohérence est un des principaux problèmes engendré par la réplication. Maintenir la cohérence entre diverses copies exige un coût important en échange des messages proportionnel au degré de cohérence offert. Dans cette section nous décrivons les deux approches utilisées dans les systèmes pour maintenir la cohérence : l'approche pessimiste et l'approche optimiste. Ensuite, nous présentons dans l'approche optimiste la cohérence incrémentale en particulier.

Cohérence pessimiste

La *cohérence pessimiste* (aussi appelée *cohérence forte*) consiste à prendre toutes les mesures de précaution pour faire face à une éventuelle incohérence. Cette approche est intéressante puisqu'elle garantit une cohérence de données tout le temps. Cependant, la garantie du maintien total de la cohérence [Lam78, Fid91, Mat88] a un coût élevé.

Le mécanisme de maintien de la cohérence est coûteux en temps et en calcul. Il existe plusieurs façons d'offrir une *cohérence forte* à l'utilisateur : (i) le contrôle atomique des transactions ; (ii) la technique de copie primaire[7] (*Read One Write All*) ; (iii) le consensus par quorum [Sou94, Tho79].

Pour assurer un contrôle atomique il faut un verrou (en écriture ou en lecture) sur chaque donnée de chaque opération (transaction). Par exemple en utilisant un algorithme de verrou à deux phases [EGLT76]. Pour que le mécanisme de verrou marche, il faut une coopération entre tous les participants du système. Le temps et les ressources utilisés pour assurer cette *cohérence forte* est parfois vain.

La technique de copie primaire [SN79] est simple en comparaison avec les autres méthodes. Un exemple populaire d'un système qui utilise cette technique est Sun NIS ou *Yellow Pages* [Wei85] qui est fourni comme un simple gestionnaire de réseau non transactionnel pour les systèmes conçus pour UNIX.

La *cohérence forte*, due à son coût, n'est pas adaptée au large échelle. Les systèmes conçus pour le large échelle travaillent avec ce que l'on appelle la *cohérence optimiste*.

7. *primary copy ROWA*

Cohérence optimiste

La *cohérence optimiste* ou *cohérence faible* fait un compromis entre le coût pour maintenir une cohérence à tout moment et le coût pour détecter et résoudre un éventuel conflit ou encore l'acceptation de certaines incohérences. Cela veut dire que le mécanisme de maintien de la cohérence marche au ralenti. Cette approche est intéressante parce qu'elle permet aux applications de fonctionner sans imposer des délais d'attente dus au protocole de cohérence.

Les mécanismes de maintien d'une cohérence faible assurent une cohérence partielle. Cela veut dire que l'ordre séquentiel des opérations de lecture et d'écriture n'est pas assuré. Dans un système à cohérence faible on trouvera par exemple une cohérence basée sur la durée de vie de la copie, sur des écritures asynchrones ou sur la vue de l'utilisateur (*cohérence incrémentale*).

Dans les caches, le relâchement de la cohérence de données est basé sur une éventuelle durée de la validité de la copie [Ter87] ; ces estimations se basent sur le moment où l'original sera mis à jour. La plupart des caches Web utilisent encore cette notion ainsi que les mises en œuvre de technologies dites *push*.

Le système de base de données Bayou [TTP$^+$95, PST$^+$97] offre une cohérence optimiste. Cette cohérence est rattachée à la possibilité qu'ont plusieurs agents d'émettre des écritures de façon asynchrone. Bayou permet à des actions de mises à jour de s'exécuter même si il faut les abandonner plus tard pour garantir la cohérence. Bayou offre aussi la possibilité de travailler dans une *session garantee*. Les *session garantee* permettent à l'utilisateur de spécifier des propriétés d'ordres de délivrance sur les lectures et les écritures.

Le système de fichiers Coda [KS92, KS91] utilise une stratégie optimiste dans le maintien de la cohérence de ses réplicats. Grâce à la cohérence optimiste, Coda offre une plus grande disponibilité de ses données permettant des lectures et des écritures même quand les serveurs et les clients sont partitionnés.

La plate-forme de développement Core proposé par [BC98] étudie la cohérence de données répliquées partagées par un groupe de processus coopérant à distance. Core n'impose pas un ordre commun unique sur l'exécution des opérations. Les réplicats sont autonomes et ne coopèrent que lorsque leur vue de l'histoire globale ne suffit plus à garantir la correction de l'application.

La *cohérence faible ou optimiste* est mieux adaptée aux systèmes à large échelle [LLG92] et aux systèmes mobiles. Son aspect asynchrone permet la mise en œuvre des protocoles pouvant être implantés sur un réseau étendu.

Cohérence incrémentale

La cohérence incrémentale représente la garantie de la *progression monotone* des opérations de lecture. Elle garantit qu'un utilisateur ne verra jamais une suite de versions d'un document revenir sur le passé. Pour que la cohérence incrémentale soit possible, un historique des versions de tous les documents fournis à l'utilisateur doit être gardé. Cela revient à maintenir une *estampille* (un seuil) qui identifie la version la plus récente de chaque document qui a été délivré à l'utilisateur. Seulement une copie du document ayant une *estampille* plus grande ou égale à ce seuil peut être donnée à l'utilisateur.

Le système de coopération de caches Relais [MPKD98] propose la cohérence incrémentale à ses clients. Pour cela, il faut faire en sorte que chaque requête d'un client n'aille pas chercher le document de version inférieure même s'il existe sur un cache voisin. Ainsi, chaque cache garde dans son répertoire le numéro de version le plus récent de chacun des documents visités par l'un de ses clients[8].

8. Tous les clients sont censés ne pas changer de cache afin que cette garantie soit respectee

Bayou [TDP+94] offre la *progression monotone* des opérations de lecture à travers une *session garantee*. Une *session* est une abstraction pour la séquence des opérations de lectures et d'écritures effectuées pendant l'exécution d'une application. Les applications peuvent choisir le niveau de cohérence voulu dans le contexte d'une *session*. Bayou implémente cette solution avec des *vecteurs de versions* (voir la section 2.3.4).

Détection et résolution des conflits

La réplication de données peut créer plusieurs possibilités d'incohérence lors de mises à jour concurrentes. Des incohérences peuvent survenir entre plusieurs utilisateurs après une déconnexion ou même quand un seul utilisateur (ou application) effectue des modifications dans plusieurs serveurs. Des mises à jour concurrentes génèrent des conflits entre les différentes copies modifiées. Il existe trois approches pour détecter et résoudre des conflits. La détection et résolution des conflits par l'utilisateur, par l'application [KS93a] et par le système.

- **Détection et résolution des conflits par l'utilisateur**

  Dans cette approche, après détection d'un conflit, les utilisateurs concernés se mettent d'accord pour résoudre le conflit. Cette méthode a l'avantage de chercher la solution la plus appropriée à un conflit. Cependant, le temps nécessaire à la gestion des conflits est élevé même si la détection est automatique.

- **Détection et résolution des conflits par l'application**

  Cette approche est basée sur le fait que les conflits ne peuvent pas toujours être identifiés en observant simplement des séquences de lectures et d'écritures. Pour faire face à ce problème, la détection et la résolution des conflits se fait en prenant en compte la sémantique de l'application.
  Le système de base de données répliquées Bayou utilise la détection et résolution des conflits par l'application [TTP+95].

- **Détection et résolution des conflits par le système**

  Dans ce cas, le système détecte et résout les conflits automatiquement. La détection des conflits est faite par une vérification de dépendances. Ces dépendances peuvent prendre en compte des caractéristiques spécifiques de l'application pour les opérations de lecture et d'écriture. Cela peut être représenté par des *vecteurs des versions* et par des *estampilles*. Cette approche a l'avantage d'être transparente à l'utilisateur et d'assurer une convergence des données. De plus, elle est nettement plus rapide que la solution manuelle.

Le protocole de résolution des conflits utilisé par Coda [Sat89, SKK+90] est capable de déterminer l'ensemble de mises à jour partitionnées qui ont eu lieu. Le protocole vérifie si il possible de résoudre ces mises à jour, effectue les mises à jours dans les sites importants et marque les données comme résolues ou incohérentes. Coda utilise différents méthodes pour la résolution des conflits sur les fichiers et sur les répertoires selon leur sémantique. La résolution pour les répertoires est basée sur des opérations gardées dans des journaux dans les serveurs [Kum94, KS93b]. La résolution pour les fichiers est faite soit par la propagation du réplicat dominant du fichier, soit en utilisant une résolution spécifique de l'application si fournit [Kum95, Kum94]. Sinon, Coda fournit un outil de résolution manuel [Ebl98, Kum94] utilisé directement par l'utilisateur.

- **Techniques de vérification des conflits**

  - **Estampilles**

    Les *estampilles* permettent l'ordonnancement des opérations de lecture et d'écriture. La *estampille* peut être un compteur (un entier) ou une date (date et heure). Pour chaque opération d'écriture effectuée, la valeur de la *estampille*, correspondante à cette donnée, est incrémentée. Cette technique est utilisée dans les études faites sur des *horloges* et sur la cohérence des données.

  - **Vecteurs de versions**

    Les *vecteurs des versions* [JPR+83] représentent l'état des objets par rapport aux opérations de lecture et d'écriture. Chaque copie de la donnée (réplicat) a son *vecteur de versions* avec la version de chaque objet. Les réplicats utilisent les *vecteurs des versions* pour comparer entre eux les versions de chaque objet. De cette façon, il est possible de détecter les réplicats avec des versions différentes. Il est alors possible de vérifier entre les versions différentes si il y a conflit.

Dans Coda [Sat89, SKK+90] les conflits en écriture sont détectés en utilisant des vecteurs des versions . Un vecteur des versions résume l'historique des modifications faites à un fichier. La taille du vecteur est le nombre des réplicats du fichier. Chaque entrée contient le nombre de mises à jour effectués au serveur correspondant.

- **Techniques de résolution des conflits**

  - **Merge**

    La technique de *merge* permet la jonction de plusieurs versions. Cela est possible grâce à une comparaison entre les modifications faites. Ces comparaisons peuvent être faites par exemple ligne-par-ligne comme est le cas dans le système CVS. Le système CVS *merges* les mises à jours faites sur un fichier par un mécanisme de comparaison ligne-par-ligne.

  - **Roll-back**

    La technique de *roll-back* est basée sur la reexécution des opérations faites. Ces opérations sont gardées dans un journal au fur et à mesure et par chaque copie. Dans le cas d'un conflit, l'application est capable de revenir sur un état cohérent grâce au mécanisme de retour en arrière sur les opérations responsables du conflit ou ayant été contaminée par le conflit.

Le système Bayou [DPS+94, TTP+95] et le système Coda [KS91] utilisent cette stratégie. Lorsqu'un conflit a lieu Bayou effectue un *roll-back*, une reordenation des opérations dans le journal et une réexécution.

2.4 Système de désignation

Le système de désignation sert à identifier un objet de sorte à pouvoir le localiser par la suite. Nous nous intéressons au système de désignation pour comprendre son fonctionnement et ses limitations. En fait, comme une des difficultés de la réplication est la localisation de ses réplicats, le système de désignation joue un rôle fondamental.

La désignation d'objets peut se faire à plusieurs niveaux. Nous pouvons désigner un objet par son nom de façon symbolique et désigner un objet par son adresse d'une manière physique. Pour atteindre l'objet lui-même, il est nécessaire d'utiliser un mécanisme de localisation (voir la section 2.5) capable de retrouver dynamiquement l'emplacement physique d'un objet à partir d'un lien symbolique (son nom).

Il est parfois difficile de distinguer une adresse physique d'un nom. Une distinction possible est que l'adresse d'un objet est temporaire et son nom est permanent.

2.4.1 Nom

Pour qu'un objet puisse se déplacer ou être répliqué durant sa vie, son nom doit être indépendant de sa localisation. La résolution à partir d'un nom indépendant de l'adresse se fait par un service de nommage capable de lier à ce nom l'adresse physique de cet objet. Cela n'est pas toujours le cas comme nous pouvons le constater dans le système de désignation par URL (Unique Ressource Locator) [URL, BLMM94].

Un nom peut donner une désignation logique à un objet et en plus indiquer sa localisation. Ce type de nom présente l'inconvénient de lier définitivement un objet à sa localisation. De ce fait, en utilisant ce type de nom l'objet ne peut pas migrer. L'avantage d'un nom lié à sa localisation est le passage à l'échelle. En fait, un espace de nommage divisé en domaines ou composé (nom de la machine + nom objet) permet une localisation d'une façon hiérarchique et supporte donc la localisation d'un grand nombre d'objets.

Les auteurs de [Nee93] distinguent les noms indépendants de la localisation de l'objet et les noms liés à la localisation de l'objet respectivement comme des *noms purs* et des *noms impurs*. Nous constatons que les noms utilisés sont normalement des noms impurs pour des raisons de performance et d'unicité de nommage.

2.4.2 Adresse

L'adresse d'un objet peut référencer plusieurs niveaux. Nous pouvons adresser l'objet par rapport au réseau où il se trouve et par rapport à la machine où il réside. De plus, si l'objet est un objet actif (en exécution), nous devons l'associer à une adresse référente au processus qui l'héberge et à son adresse mémoire.

Un objet qui existe en plusieurs copies a plusieurs adresses physiques. L'utilisateur voulant accéder cet objet doit connaître au moins une de ces adresses. Un objet qui se déplace change d'adresse durant sa vie. Dans ce cas, l'adresse est une information temporaire. Elle permet de trouver l'objet à un instant donné.

2.4.3 Espace de nommage

- **Plat**
  Dans un espace de nommage plat, tous les noms sont formés d'une composante. La résolution, dans ce cas, peut se faire directement via une table de correspondance contenant de couples (*nom, localisation*).
  Lorsque l'espace de nommage est plat, il est difficile de garantir que chaque nom sera différent de l'autre. Il est encore plus difficile d'assurer l'unicité du nommage dans le cas d'un grand réseau. Lorsque plusieurs autorités administratives peuvent nommer en toute indépendance, il est invraisemblable qu'elles puissent donner que des noms différents.

- **Hiérarchique**
  Dans le cas des réseaux à large échelle, le nommage utilise plusieurs espaces de nommage. Les différents espaces de nommage sont organisés de façon hiérarchique. Chaque autorité administrative est responsable de son espace de nommage et peut nommer ses objets en toute indépendance.
  L'espace de nommage du système DNS [Moc87] est organisé sous la forme d'un arbre hiérarchique où chaque nœud de cet arbre correspond à un nom de domaine. DNS propose une vue globale du système.
  Dans Globe [HST96, SHTH98], l'univers est divisé hiérarchiquement dans un ensemble des domaines. Dans les feuilles de cet arbre on rencontre les sites proprement dits. Un ensemble des sites forment une région, etc. Un objet est enregistré dans chaque site où il a une adresse, et d'une façon récursive pour chaque région englobant jusqu'à la racine de l'arbre.
  Grapevine [BLNS82] utilise un espace de nommage hiérarchique à deux niveaux pour les RNames. Un RName est de la forme F :R, où R est un répertoire de noms d'une partition et F un nom dans ce dernier.

- **Décentralisé**
  Un espace de nommage décentralisé utilise un entrepôt global réparti sur différents gestionnaires d'objets. Ces derniers sont responsables chacun de leur espace de nommage (i.e les objets sur les nœuds de leur domaine).
  Le System V [CM86] a été conçu avec un approche décentralisé de nommage. L'espace de nommage consiste en un entrepôt global contenant les correspondances entre les noms des objets et leurs références.

2.5 Localisation et découverte dans le large échelle

2.5.1 Annuaires

Les annuaires sont des services capables d'associer à un nom d'objet, une localisation physique.

La désignation d'un objet a pour objectif de permettre son accès. Dans ce sens, le système de désignation doit fournir un service capable de mettre en relation un nom avec l'objet correspondant.

La résolution d'un nom manipulé par l'utilisateur (chaîne de caractères) se fait, en général, via un service de nommage utilisant des annuaires. Lorsqu'un client veut retrouver un objet serveur, il interroge le service de nommage en lui fournissant le nom de l'objet recherché. Lorsqu'un nouvel objet serveur est crée, il doit se faire connaître au service de nommage. Une autre façon de désigner un objet est de lui attribuer une série des mots-clés qui caractérisent le contenu de cet objet et ses intérêts.

La localisation consiste à retrouver dynamiquement l'adresse d'une entité à partir de son nom, d'un mot-clé ou même d'un adresse dans le cas des réplicats.

Dans le cas des objets répliqués, certains systèmes de localisation considèrent d'autres critères que le nom dans la mise en relation. Ces critères prennent en compte la valeur de l'objet, son emplacement ou sa distance.

Serveurs de noms

Cette stratégie consiste à stocker sur une ou plusieurs machines des correspondances (*nom, localisation*). Les serveurs de noms retournent pour chaque entité recherchée, l'adresse IP de la machine hôte. À chaque requête, les clients peuvent conserver dans un cache les informations transmises par le serveur de nommage.

Le risque de cette stratégie est de stocker des informations incohérentes. Les informations doivent être mises à jour, ce qui représente un coût variable en fonction de la durée de vie des objets et du taux de migration des objets.

Le serveurs de noms peuvent être répartis comme est le cas dans le système DNS [Moc87]. Plusieurs machines hébergent des serveurs de nom qui connaissent chacun une partie des entités nommées sur Internet.

Bases de données de noms distribuées

Les bases de données de noms distribuées, différent des serveurs de noms par le fait qu'elles sont gérées par des langages de requêtes spécifiques. Ces bases de données peuvent avoir des entrées plus complexes. L'aspect distribué permet à la fois de mieux partager la charge et aussi d'offrir une meilleure disponibilité.

Grapevine maintient une base de données de nommage (*registration data base*) qui lie un nom aux informations qui le concernent. Il y a deux types d'entrée dans cette base de données : individuelle et de groupe. Une entrée dans la base est appelée RName. Un groupe correspond à un ensemble de RNames (groupe ou individuelle). Une entrée individuelle contient, entre autres, un mot de passe, une liste des sites contenant un réplicat de la boîte de messages et un site de connexion (il correspond au site de l'individu).

Le système Clearinghouse [DCYK83] utilise aussi des bases de données de noms distribuées pour gérer son système de nommage.

Répertoires répliqués

L'utilisation des répertoires répliqués consiste à avoir pour chaque agent du système une copie du contenu des objets connus des autres entités du système au niveau répertoire. Chaque répertoire local connaît les objets présents localement et les objets présents chez les autres agents.

Le système Relais [MPKD98] garde un répertoire contenant la localisation de la totalité des documents se trouvant dans les caches voisins. Ainsi, si le document demandé n'est pas dans le cache local, ce dernier regarde dans son répertoire pour voir si ce document se trouve dans un cache voisin ou pas. Si oui, il demande le document au cache voisin et le retourne au client, sinon il fait la requête directement au serveur source de ce document.

2.5.2 Moteurs de recherche

Les moteurs de recherche proposent la découverte d'informations par comparaison des mots-clés. Le résultat des milliers de comparaisons retourne une liste des documents possibles au client ayant fait la requête. L'avantage de cette technique est le côté autonome de la recherche. À la place de dépendre d'un système ou d'autres entités pour connaître la localisation des données, les moteurs de recherche découvrent par eux-mêmes leur existence et leur localisation .

Cependant, les moteurs de recherche ne garantissent pas la localisation d'un document spécifique. Ils sont donc plus adapté à la recherche d'une information au sens large.

Il existe de nombreux moteurs de recherche largement divulgué et utilisé par des nombreux utilisateurs tels que : AltaVista [Alt], Yahoo [Yah] et Voilà [Voi], Google [Goo], alltheweb [all], etc.

2.5.3 Les systèmes de nommage classiques

Dans cette section nous présentons les solutions classiques proposées par certains systèmes de nommage.

Clearinghouse

Clearinghouse[9] [DCYK83] est un service de nommage conçu à Xerox. Il adresse les problèmes de nommage d'objets et leur localisation dans un système réparti.

Le nommage dans Clearinghouse est hiérarchique à trois niveaux : *L:D:O*, où *L* correspond au nom local de l'objet, *D* son domaine et *O* son organisme. En fait, ce n'est qu'un DNS à trois niveaux.

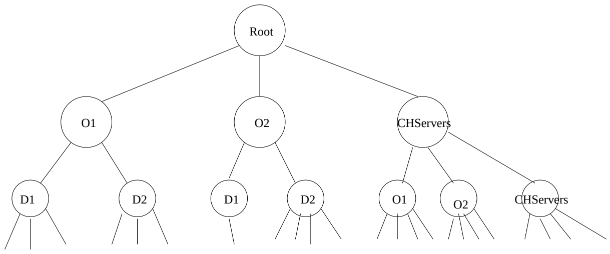

FIG. 2.1 – *Espace de nommage dans Clearinghouse*

Un nom sous Clearinghouse se présente sous la forme *Nom:Domaine:Organisme*. Un nom à l'intérieur d'un domaine doit être unique.

Clearinghouse associe à chaque nom en ensemble de propriétés. Chaque propriété est une tuple contenant un *nom de propriété*, un *type de propriété* et une *valeur de propriété*.

```
Neilze Dorta:Doctorante:SOR
{
  (Ordinateur, hobbes.inria.fr, xxx.xxx.xxx.xxx),
  (Imprimante, mops.inria.fr, yyy.yyy.yyy.yyy),
}
```

9. cela veut dire, une agence pour la collection et la distribution d'informations

Les serveurs de nommage de Clearinghouse sont répartis. Chaque serveur, ou chaque ensemble de serveurs, contiennent une partie du répertoire de nommage global. Chaque client utilise un *stub* qui a la charge de contacter le service de nommage de Clearinghouse et de retourner la réponse au client.

A chaque domaine D dans un organisme O correspond un ou plusieurs serveurs de noms, chacun contenant une copie des mappings des objets du domaine (dont le nom se termine par D :O). Il y a au moins un serveur par domaine. Or les serveurs de domaines sont aussi des objets addressables et ont ainsi un nom. Chaque serveur de domaine D dans un organisme O a un nom de la forme <nom> :O :CHServers. Cette entrée dans l'espace de nommage correspond à l'adresse réseau de ce serveur. Chaque domaine D de l'organisme O a obligatoirement une entrée dans l'espace de nommage D :O :CHServers qui permet de retrouver la liste de tous les serveurs de ce domaine. Les O :CHServers sont des domaines aussi adressables. Il leur faut alors un serveur de noms qui va gérer leur espace de nommage. Chaque serveur de nom d'un organisme O aura un nom de la forme <nom> :CHServers :CHServers. Il pointera sur un serveur de nom d'un des organismes. Le nom O :CHServeers :CHServers correspondra à la liste des serveurs de noms de l'organisme O.

La liste des serveurs de noms des organismes (l'espace de nommage CHServers :CHServers) est répliquée dans chaque serveur de noms de domaine. on suppose que tout stub lié à une machine pointe vers un serveur de noms.

Lors d'une requête vers un objet x :B :C, le stub local contacte le serveur de nom auquel il est lié, Si ce dernier est un serveur du domaine B :C, il répond immédiatement à la requête du client. Sinon, il retourne la liste des serveurs de noms de l'organisme C qui contiennent la liste des noms des serveurs de domaine B. Le stub contacte alors l'un de ces serveurs. S'il est le serveur de domaine de B :C il répond directement, sinon il retourne la liste des serveurs du domaine B :C. Le stub contacte alors un de ces serveurs et reçoit la réponse. Au pire des cas, le stub aura à contacter trois serveurs avant d'avoir sa réponse.

Plusieurs méthodes sont utilisées pour écourter ce processus. Le premier étant d'utiliser un cache afin de résoudre rapidement les noms qui ont déjà été résolus auparavant. Une deuxième méthode serait de laisser une entrée dans un serveur de domaine pointant sur un autre serveur de domaine fréquemment utilisé.

La mise à jour des serveurs de noms répliqués se fait d'une manière asynchrone. Le client, à travers son stub, contacte le serveur de nom du domaine, et lui soumet un changement. Ce dernier acquitte ce changement et envoie les mises à jour d'une manière asynchrone aux autres serveurs du domaine. Ainsi, il pourrait y avoir entre temps des erreurs de nommage. Or, ceci n'est pas très grave puisque les liens faux seront vérifiés faux au moment où le client essaiera de les utiliser. C'est pour cette raison que les liens fournis sont appelés des « hints » .

Grapevine

Grapevine [BLNS82] est un système développé dans le début des années 80 chez Xerox. Il fournit des services de transport de messages, de localisation de ressources, d'authentification et de contrôle d'accès.

Grapevine maintient une base de données de nommage (registration data base) qui lie un nom aux informations qui le concernent. Il y a deux types d'entrée dans cette base de donnée : individuelle et de groupe. Une entrée dans la base est appelée RName. Un groupe correspond à un ensemble de RNames (groupe ou individuelle). Une entrée individuelle contient, entre autres, un mot de passe, une liste des sites contenant un réplicat de la boîte de messages, un site de connexion (il correspond au site de l'individu).

Grapevine utilise un espace de nommage hiérarchique à deux niveaux pour les RNames. Un RName est de la forme F :R, où R est un répertoire de noms d'une partition et F un nom dans ce dernier.

Dans Grapevine, il y a deux entités différentes : les serveurs de noms (registration servers) et les serveurs de messages (message servers). Les serveurs de noms mettent en oeuvre la localisation. Dans ce qui suit, on va focaliser sur cette partie de Grapevine.

La base de donnée de noms est distribuée et répliquée. La distribution se fait au niveau des partitions. De même, chaque partition est répliquée dans plusieurs serveurs de noms. Les mises à jour sur les réplicats se font de manière asynchrone et les conflits possibles résolus par les serveurs.

Si un utilisateur nommé P.Q, utilisant une machine 1, veut envoyer un message à X.Y, alors : (1) il contacte un serveur de noms pour localiser une serveur de messages B. (2) Puis, il envoie le message à B. Pour chacun des récipients du message, (3) B utilise les fonctions de localisation de Grapevine pour déterminer le meilleur serveur de messages contenant la boîte de message correspondant à chacun des récipients. (4) Pour lire ces messages, X.Y (5) utilise le serveur de noms pour déterminer les serveurs de noms responsables pour Y, puis (6) contacte chacun des sites contenant une boîte de message du client pour retirer son contenu.

L'espace de nommage dans Grapevine est divisé en domaines correspondant chacun à un service : le domaine des serveurs de noms (gv), le domaine des serveurs de message (ms), ...

Le domaine des serveurs de noms est répliqué sur tous les serveurs de noms. Chaque serveur de nom d'une partition est représenté dans l'espace de nommage par une entrée individuelle. Une entrée de groupe est aussi associée à chaque partition et contient la liste des noms des serveurs de cette dernière. Ainsi, pour retrouver la liste des serveurs de noms de f.r, il suffit de se connecter sur n'importe quel serveur de noms et demander la liste des serveurs de l'entrée r.gv. Il reste le problème de trouver le premier serveur à contacter. Pour cela, Grapevine utilise soit un serveur de noms local connaissant un ensemble de serveurs de noms, soit il « broadcast » un message sur le réseau local en espérant avoir une réponse d'un serveur de noms local (ces deux méthodes peuvent être utilisées simultanément pour limiter le temps de réponse).

La mise à jour des serveurs de noms se fait d'une manière « best effort » sans garanties. Grapevine assure qu'après un certain temps tous les serveurs de noms d'une partition convergent vers une même identité.

System V

Le nommage sous SystemV

SystemV est un système d'exploitation réparti qui a été développé par l'équipe de Cheriton à l'université de Stanford. Dans cette section, on s'attaquera seulement au système de nommage des objets offert par ce système [CM86].

SystemV implémente une approche décentralisé de nommage. L'espace de nommage est vu comme un entrepôt global partitionné sur plusieurs noeuds (voir figure 2.2). Chaque client maintient un cache local de noms pour réduire le trafic réseau ; un miss induit alors un *multicast* au groupe de serveurs de noms.

Conception

Conceptuellement, le nommage consiste en un entrepôt global contenant les correspondances entre les noms des objets et leurs références (nom, référence). Chaque client peut avoir sa propre table de nommage des noms locaux (ou aliases) vers les noms globaux. Le mécanisme de nommage permet de :

- relier un objet à sa référence,
- supprimer des références
- trouver les référence liées à un nom
- trouver le nom lié à une référence (inverse name mapping)

Le système de nommage étant décentralisé, le entrepôt global est réparti sur différents managers d'objets (voir figure 2.2). Ces derniers sont responsables chacun de son espace de nommage, i.e les objets sur son noeud.

Lorsqu'un client invoque une opération utilisant le nom d'un objet, il regarde dans son cache pour voir si la résolution a déjà été faite. Si oui, l'opération sur le nom est "forwardé " au manager correspondant indiqué par l'entrée dans le cache. Sinon, une demande est envoyée à tous les managers d'objets pour leur demander l'identité du manager de l'objet nommé. Si un de ces managers d'objets répond, une entrée dans le cache local est créée, sinon, le nom spécifié est considéré invalide et une erreur est retournée au client.

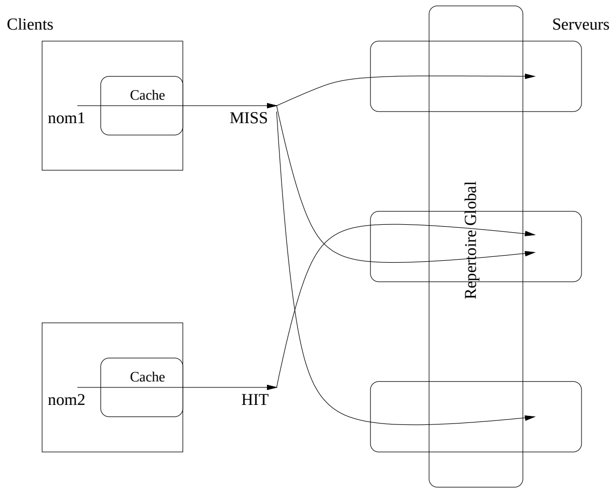

FIG. 2.2 – *System V*

Une requête de *inverse name mapping* correspond à une recherche dans la partie du entrepôt liée à un manager d'objet. La référence d'un objet identifie son manager et ainsi le client envoie la demande à ce dernier qui est censé retourner le nom de cet objet. La partie lié au manager de l'objet n'est pas identique pour tous les managers ; ainsi chaque manager peut avoir sa propre méthode de nommage interne.

La conception de ce système de nommage colle à l'implémentation des mémoires partagées pour des multiprocesseurs. Les avantages d'une telle architecture sont les suivantes :

Efficacité : Avec un pourcentage élevé de hits, le temps d'accès au entrepôt global est rapide,

disponibilité : Un manager d'objets en panne met hors service le service de nommage d'une partie seulement du entrepôt global,

Extensibilité : Le entrepôt global peut être étendu par le simple ajout de nouveaux managers d'objets.

Cohérence des caches La cohérence d'un cache client est maintenu au moment de l'utilisation de l'objet (On-Use consistency). (1) Si une entrée du cache existe déjà, le manager de l'objet est contacté pour savoir si l'entrée du cache est valide ou pas. (2) Si le manager de l'objet n'existe plus ou est en panne, le système notifie au client l'erreur au moment où ce dernier tente de contacter le manager. (3) Si le manager existe mais n'est plus responsable de cet objet, il retourne une notification d'erreur au client. Dans les deux derniers cas, l'entrée dans le cache est supprimée et une requête est envoyée à l'ensemble des managers d'objets pour savoir si l'objet existe encore et qui en est son manager.

Gestion des alias Les alias ne sont rien d'autre que des entrées dans le cache local pointant sur une autre entrée dans le cache (qui peut être aussi un alias) et qui pointe éventuellement, en suivant les pointeurs, sur une entrée contenant un nom et une référence dans le entrepôt global.

2.6 Support pour la localisation à large échelle

Dans les systèmes informatiques multi-utilisateurs, il est fondamental de nommer les différentes machines de façon unique. Ceci afin de pouvoir accéder les objets de chaque machine en utilisant leur nom associé au nom de la machine où l'objet se trouve et ne pas avoir d'ambiguïté. Dans les systèmes à large échelle, cette tâche devient plus compliquée, étant donnée le grand nombre des machines et donc des noms à connaître.

2.6.1 DNS (*Domain Name System*)

DNS (*The Domain Name System*) [Moc87] est un système de nommage. Il assure que chaque machine a un nom unique dans le réseau. En outre DNS fournit la correspondance entre un nom de machine et son adresse.

L'espace de nommage du système DNS est organisé sous la forme d'un arbre hiérarchique. Chaque nœud de cet arbre correspond à un nom de domaine composé de sa propre étiquette associée à la liste des étiquettes des nœuds qui sont ses ascendants. Tous les nœuds dans l'arbre doivent avoir un nom de domaine unique, mais l'étiquette d'un nœud peut être utilisée dans plusieurs endroits différents de l'arbre. La figure 1 montre la hiérarchie de ce système de nommage.

Le lien entre l'adresse IP et le nom d'une machine doit être créée et stocké pour les serveurs de noms. L'interface (*resolver*) utilisé en Unix pour obtenir le lien est dans deux fonctions de librairie : ***gethostbyname()*** et ***gethostbyaddr()***. C'est donc dans le coté client qui le *resolver* contacte le serveur de nom pour obtenir l'adresse IP avant que la connexion TCP puisse être établie.

Le qsystème DNS est une base de données distribuées. Le terme distribué est justifié puisqu'aucun site isolé est en possession de toutes les informations. Chaque site maintien sa propre base de données et maintien un serveur (serveur de nom) qui peut être interrogé par d'autres systèmes (clients) à travers l'Internet. Le DNS fournit le protocole qui permet la communication entre les clients et les serveurs.

Une caractéristique importante dans le DNS est l'attribution de responsabilité dans l'espace de nommage. Aucune entité gère les informations de l'arbre entièrement. Chaque entité maintient les informations d'une partie de l'arbre, soit une *zone* (un sous arbre de l'arbre DNS géré séparément). C'est à l'administrateur de chaque *zone* d'allouer les adresses IP qui lui appartiennent et de les associer à un nom. Un ou plusieurs serveurs de nom sont alors créées et répondront à des requêtes concernant cette *zone*.

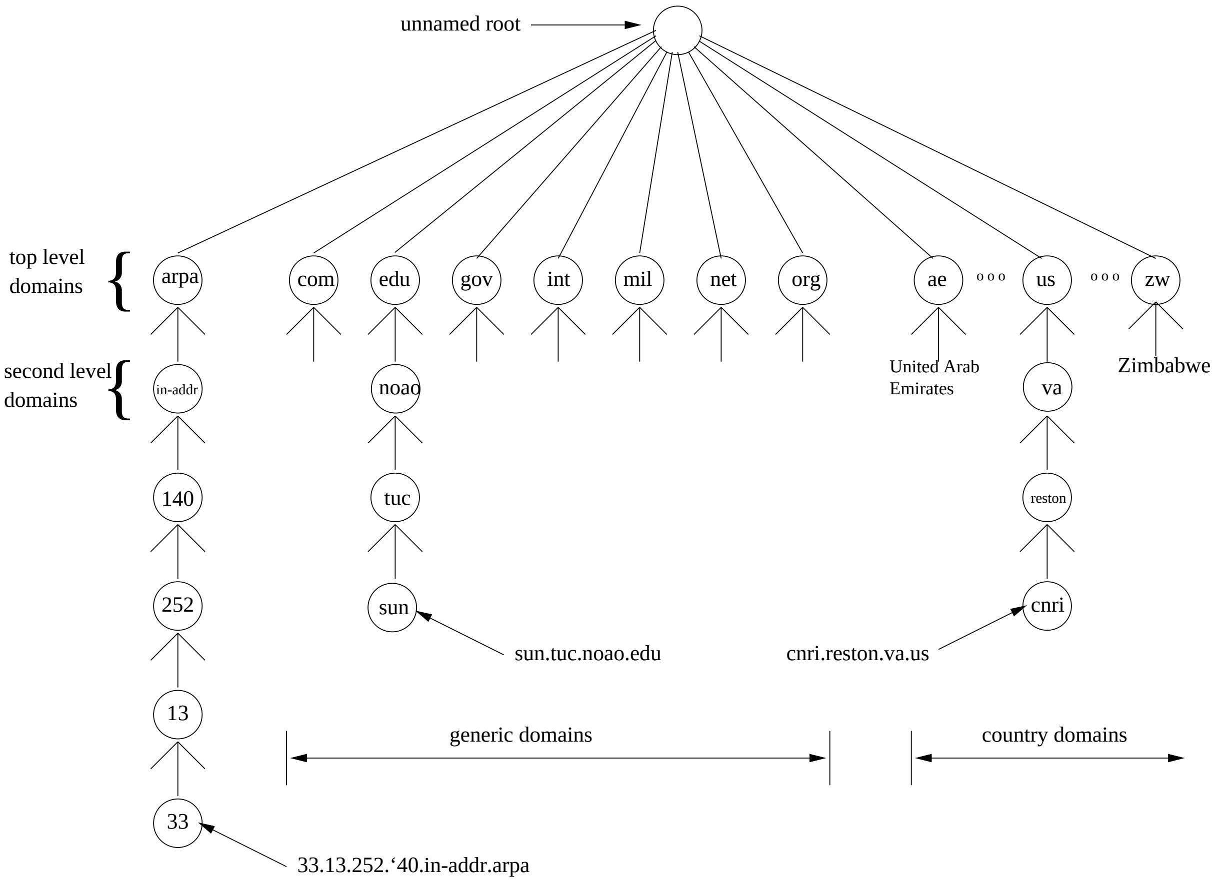

FIG. 2.3 – *Organisation hiérarchique de DNS.*

Un serveur de nom qui n'a pas l'information nécessaire pour répondre à une requête, va contacter un autre serveur de nom. Cela est propre à la nature distribuée du système DNS qui contactera la racine (capable de directionner la requête au niveau suivant de l'administration) de l'arbre au cas où aucun autre serveur ne connaît l'information.

Les requêtes DNS génèrent de trafic sur le réseau. Pour réduire ce trafic, tous les serveurs de nom utilisent un cache. Quand un serveur de nom reçoit une nouvelle information sur une correspondance (adresse IP, nom machine), il la garde dans son cache. Ainsi, lors d'une prochaine requête sur la même machine, le serveur trouvera cette information dans le cache, évitant des requêtes vers d'autres serveurs de nom.

Le système de nommage DNS résout en partie le problème de localisation rencontré sur Internet. Toutefois, pour faire face aux problèmes du large échelle, on doit répliquer des documents. Un même document finit par se trouver dans plusieurs endroits avec des adresses IP et des noms différents. On aimerait avoir un système de nommage sur Internet qui sache identifier ces différentes localisations comme étant la localisation d'un document unique.

2.6.2 Globe

Globe [SHT97, HST96, SHTH98] est un système distribué pour le large échelle, implémenté comme une couche au-dessus des réseaux existants et des systèmes opérationnels. L'architecture de Globe consiste en un modèle à objet et un support à des services de base. Le modèle à objets vise le partage d'objets dans le large échelle. Ces objets peuvent être partagés par un grand nombre des processus. Les services de base supportent entre autres le nommage et la localisation d'objet ; toujours dans le cadre large échelle.

Un objet Globe est distribué physiquement et l'état de cet objet peut être partitionné et répliqué dans plusieurs machines au même temps. Pour faire face aux problèmes dû au large échelle, le modèle à objets de Globe est basé sur les *objets distribués partagés*. Un *objet distribué partagé* est composé par des objets locaux situés dans des espaces d'adressages différents. Un objet local cache à l'utilisateur la réplication, la distribution et la migration de l'*objet distribué partagé*. Chaque objet local a sa propre interface et est composé au minimum de quatre objets : (i) l'objet sémantique, (ii) l'objet communication, (iii) l'objet réplication, (iv) l'objet contrôle. Cela garantit une flexibilité d'implémentation de chaque composant pour les objets locaux formant un seul *objet distribué partagé*. Autre avantage est la transparence, puisque l'utilisateur voit un *objet distribué partagé* comme une seule entité. Toute la partie des protocoles de communication, stratégies de réplication, distribution et migration de l'état de l'objet font partie de l'objet lui même et sont cachés par son interface.

Le système Globe associe (*bind*) un identificateur de document à un ensemble d'adresses. Cela permet la localisation des réplicats d'une façon transparente. Globe utilise un nommage à deux niveaux hiérarchiques. Le serveur de noms retourne un descripteur (*handle*) d'objet, qui est indépendant de sa localisation et qui est un identificateur d'objet unique. Ce descripteur est passé à un serveur de localisation qui retourne une ou plusieurs adresses correspondantes.

Les étapes d'un *bind* sont : (i) recherche du nom, (ii) recherche de la localisation, (iii) choisir un adresse qui convient et (iv) choisir une implémentation qui convient pour l'interface de l'objet. Le service de localisation doit pouvoir exécuter ses étapes dans le cadre du large échelle. Cela est fait par le biais d'un arbre universel de recherche distribué. L'univers est divisé hiérarchiquement dans un ensemble des domaines. Dans les feuilles de cet arbre on rencontre les sites proprement dits. Un ensemble des sites forment une région, etc. Un objet est enregistré dans chaque site où il a une adresse, et d'une façon récursive pour chaque région englobant jusqu'à la racine de l'arbre.

Globe a des caractéristiques qui permet de supporter des applications à large échelle : flexibilité dans les politiques, nommage qui supporte la réplication, hiérarchisation de l'information de localisation. Cependant, Globe tient compte d'un *rebind* de tous les objets existants, ce qui est très contraignant.

2.6.3 Caches coopératifs

Les caches coopératifs offrent un support pour la localisation des documents qui se trouvent dans des caches géographiquement proches.

Harvest

Harvest [CDN+95] est un système de caches coopératifs hiérarchiques. Chaque cache coopère avec des partenaires : voisins ou parents. Un voisin est typiquement un cache géographiquement à proximité. Un parent est un cache de niveau supérieur

Dans le cas d'une requête, si le document demandé se trouve dans le cache local, il est servi. Sinon, la requête est diffusée aux caches voisins et parents. Le premier à répondre positivement est choisi pour servir la requête, sinon la requête est redirigée vers le serveur d'origine. Les caches voisins étant géographiquement plus proches, vont répondre normalement plus rapidement à cette requête.

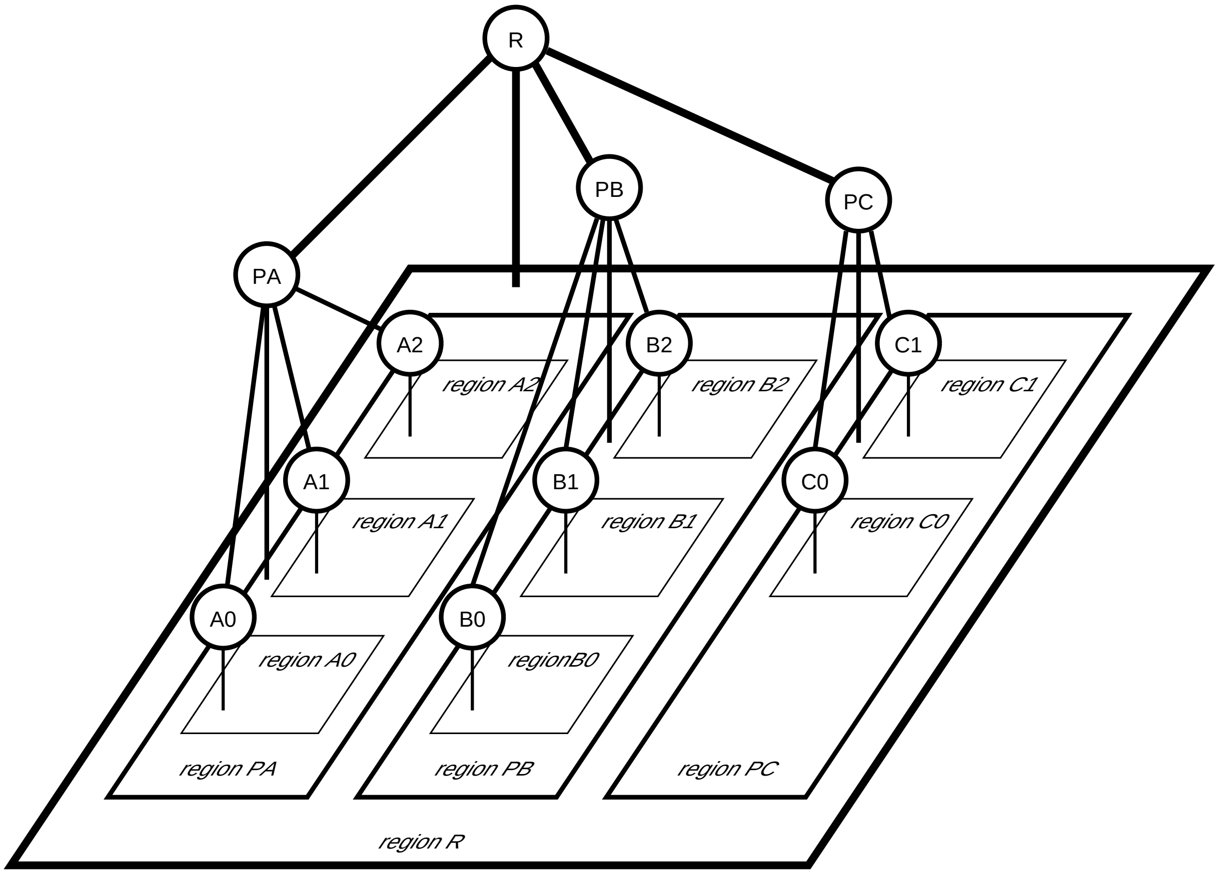

FIG. 2.4 – *L'organisation logique du service de localisation comme un arbre de recherche virtuel*

Cette méthode de localisation permet de savoir d'une manière assez sûre si un document existe ou pas dans le cache. Cependant, elle est trop gourmande en terme de bande passante. Le point le plus positif est dans l'architecture en hiérarchie des caches coopérants qui permet à Harvest de passer à l'échelle. En fait, on pourrait utiliser la même architecture hiérarchique pour Relais si on omet la garantie de cohérence.

Relais

Relais [MPKD98] est un système de localisation de documents sur le Web garantissant une certaine cohérence des données.

Relais se base sur (1) l'hypothèse de localité entre organismes et (2) sur le fait que la recherche d'un document chez un organisme coopérant est plus rapide (en terme de latence). Ainsi, propose-t-il de placer des caches dans des organismes de même centres d'intérêts afin de profiter de la localité des accès pour servir plus rapidement les documents Web.

Une requête est tout d'abord soumise au cache de l'organisme auquel elle appartient. Si le document est déjà dans le cache, il est servi. Sinon, ce cache, et d'une façon totalement transparente au client, se charge de retrouver le document et de le retourner. Pour cela, chaque cache *Relais* garde un répertoire contenant la localisation de la totalité des documents se trouvant dans les caches voisins. Ainsi, si le document demandé n'est pas dans le cache local, ce dernier regarde dans son répertoire pour voir si ce document se trouve dans un cache voisin ou pas.

Si oui, il demande le document au cache voisin et le retourne au client, sinon il fait la requête directement au serveur de ce document.

La question qui se pose maintenant est comment ce répertoire est mis à jour. Tout changement dans l'un des caches coopérants est multicasté à tous les autres. Par exemple, après l'ajout d'un document dans un des caches, ceci est notifié, de manière asynchrone, aux autres caches qui ajoutent dans leurs répertoires que ce dernier contient le document. Dans le cas d'un retrait de document, une notification est envoyée aux autres caches pour éviter qu'ils viennent demander le document alors qu'il a été évincé.

Du fait de l'asynchronisme des notifications, la vue d'un cache (son répertoire) n'est jamais totalement correcte. Ceci induit des cas où un document existe déjà dans un des caches voisins alors que la notification n'a pas été encore reçue, ce qui fait qu'une requête vers ce même document aboutit sur le serveur d'origine. De même, si un document a été retiré du cache et que cette notification n'a pas été encore reçue, une requête vers ce document aboutira sur le cache qui a déjà évincé le document ; ce dernier répond alors négativement au cache voisin qui ira chercher le document sur le serveur d'origine.

Relais propose aussi de garantir aux clients une progression monotone des versions des documents qu'ils demandent. Pour cela, il faut faire de sorte que chaque requête d'un client n'aille pas chercher le document de version inférieure même s'il existe sur un cache voisin. Pour cela, chaque cache garde dans son répertoire le numéro de version le plus récent de chacun des documents visités par l'un de ses clients[10]. Pour retirer une entrée de ce répertoire, Relais s'assure que tous les autres caches ont déjà été notifiés au minimum de la version locale du document afin d'être sûr de ne pas recevoir ultérieurement une version antérieure à la sienne. En fait, à la réception d'une notification, la version du document notifié est comparée à celle de l'entrée dans le répertoire. Si cette dernière est plus récente, la notification est rejetée. Si elle est plus ancienne, elle est remplacée. De plus, si le cache contenait localement le document, ce dernier est évincé puisqu'un cache voisin détient maintenant une version plus récente. Cette dernière action permet de faire profiter le plus vite possible l'ensemble des coopérants, et ainsi leurs clients, des versions les plus récentes des documents. Cette garantie de cohérence impose une architecture plate de coopération, ce qui fait que Relais ne passe pas à l'échelle.

2.7 Systèmes pour la localisation des miroirs

La localisation des miroirs s'avère importante pour améliorer la performance, tant du point de vue de temps d'accès comme de diminution de la charge réseau en nombre des messages. Nous présentons des études faites récemment dans ce domaine. Chacune de ces études propose une technique différente : découvrir les miroirs par une sorte de « moteur de recherche » ou adapter le DNS pour traiter les réplicats.

2.7.1 Localisation automatique des réplicats

Les travaux faits par [APS98] proposent trois solutions pour sélectionner de façon transparente des réplicats de serveurs Web. Une méthode de redirection via HTTP, une méthode de redirection via DNS et une méthode de partage d'adresses IP au niveau du routage du réseau. Les deux dernières utilisent *DNS Update* [VtYB97] afin de modifier une adresse IP liée à un serveur.

10. Tous les clients sont censés ne pas changer de cache afin que cette garantie soit respectée

La méthode HTTP

Cette méthode est implémenté utilisant la programmation du côté du serveur au niveau de l'application. L'application est à un niveau où la topologie du réseau n'est pas connue ainsi que la localisation du client dans le réseau. La redirection de la requête via HTTP connaît la meilleur copie entre toutes les réplicats mais ne peut pas relativiser cette donnée par rapport à chaque client en particulier.

Un point positif de cette méthode est la possibilité de prendre en considération le contenu de la requête avant de choisir le « meilleur » serveur à accéder. Il est possible de déterminer qu'un serveur soit mieux dans un certain service (par exemple un serveur optimisé pour exécuter de CGI).

La méthode DNS

Cette méthode utilise le système DNS [Moc87, VtYB97] pour aider dans la détermination du serveur le plus proche du client. La méthode tire profit du fait que les serveurs à chaque requête vers un serveur distant, gardent le temps d'aller-retour (*RTT - round trip time*) des paquets vers ce serveur. À la longue, le serveur local va favoriser les serveurs qui répondent plus vite. Cela fait un bon système pour connaître la localisation du client par rapport au serveur.

L'idée principale de cette méthode est de lier à chaque serveur de nom (au sens DNS) l'adresse IP du réplicat le plus proche de ce serveur de nom. Puisque le client finira par accéder au serveur de nom le plus rapide, il finira aussi par accéder au réplicat le plus rapide. Ce système est possible grâce à l'utilisation de *DNS Update* qui permet d'affecter chaque serveur de nom avec la copie du serveur Web vers laquelle on veut diriger les utilisateurs de cette région particulière du réseau

Cette méthode a l'avantage de n'avoir pas besoin des modifications sur l'infrastructure du réseau pour fonctionner. Cependant, on note que l'optimisation offerte par DNS peut être une bonne idée pour savoir la proximité du serveur du nom par rapport au client, mais peut être ça ne donne pas d'indication sur la proximité du client par rapport au serveur Web.

Le principal désavantage de cette méthode est le temps nécessaire pour établir quel serveur de nom est le plus rapide et pouvoir donc accéder la copie proposé par ce serveur. Cette méthode est donc intéressante pour un usage à longue terme et ne doit pas bien fonctionner pendant une courte période de temps.

La méthode de partage des adresses IP

Normalement une adresse IP dans l'Internet correspond à une machine. Dans cette méthode, on propose d'affecter une seule adresse IP à plusieurs machines. Le partage d'une adresse IP par plusieurs machines sur un réseau à large échelle permet une distribution intelligente de la charge réseau.

L'objectif est celui de partager des adresses IP et de savoir localiser l'instance la plus proche d'un adresse IP. En effet, cela donnerait l'illusion qu'une machine logique est connectée dans plusieurs endroits du réseau via une connexion très rapide. Dans la pratique, on a un réplicat différent pour chaque connexion.

Comme dans la méthode DNS, les adresses différentes, qui sont liées à une adresse IP unique, sont affectées aux serveurs DNS. Chaque serveur DNS est affecté avec l'unique adresse IP du serveur Web le plus proche de lui. C'est grâce à l'utilisation de *DNS Update* qui chaque serveur de nom peut être affecté avec une adresse IP différente pour le même serveur Web.

Nous ne voyons pas une vraie différence avec la méthode DNS si ce ne que dans l'abstraction d'une seule adresse IP pour plusieurs machines.

2.7.2 LDS

LDS est une extension du système DNS (Domain Name System). LDS devrait permettre un serveur quelconque d'obtenir l'adresse IP des serveurs qui stockent une URL spécifique.

Le travail de [KRR99] propose un service de correspondance. Un client donne une URL au système LDS qui retourne une liste des serveurs qui possèdent l'objet. LDS suppose certaines modifications dans le système de *binding* de DNS.

Comme DNS, LDS utilise une hiérarchie des serveurs pour implémenter une base de données distribuées avec des registres des ressources. Chaque ressource LDS associe une URL vers un serveur. Chaque URL contient un (ou plusieurs) serveur faisant autorité de localisation. Le serveur faisant autorité de localisation contient une liste de registres de ressources pour des URL, c'est à dire, une liste de serveurs qui possèdent l'URL ainsi que leur mis à jour.

LDS fonctionne de la façon suivante : le serveur demandeur émet une requête de localisation vers son serveur de localisation local. Si le serveur de localisation n'a pas l'information de localisation dans le cache, il envoie une requête vers un serveur *root* de localisation. Si ce dernier n'a pas l'information dans le cache, il retourne l'adresse d'un serveur responsable du domaine du serveur d'origine de la requête. Le serveur demandeur émet une nouvelle requête au serveur responsable du domaine. Si celui-ci n'a toujours pas l'information, il retourne l'adresse du serveur faisant autorité de localisation. Enfin, le serveur de localisation local émet une requête au serveur faisant autorité de localisation pour recevoir l'information de localisation. L'information est alors envoyée au serveur demandeur.

LDS augmente de façon considérable le nombre des messages envoyés sur le réseau et peut causer une surcharge des serveurs et des liens dégradant ainsi la performance. LDS est donc peu envisageable pour localiser des documents dans des caches. En ce qui concerne la localisation des miroirs, LDS pourrait être plus approprié. LDS serait donc une solution globale avec un coût correspondant aux dimensions de l'Internet. LDS n'a pas été implémenté et donc nous n'avons aucune mesure disponible.

Kangasharju a aussi présenté un étude [KRR00] d'évaluation de la performance dans différents schémas de redirection d'une requête vers un réplicat.

2.7.3 Localisation des miroirs basée sur la topologie du réseau

La solution présentée dans [Pul98] implémente une méthode qui utilise l'information de routage disponible, provenant des sources qui fournissent les informations concernant les chemins des systèmes autonomes (comme le *Internet Routing Register (IRR)*) pour déterminer, en principe, un miroir « proche ». Le miroir « proche » est choisi comme étant celui qui est atteignable en traversant le nombre minimum des systèmes autonomes (SA) pour toutes les adresses IP. Avec ces informations une table de « mirorage » se construit. Le miroir est choisi en consultant cette table.

La table de « mirorage » est créée par les informations contenues dans l'*IRR* sur les routeurs réseau et par les annonces qui les *ISPs* veulent envoyer et recevoir d'autres *ISPs*.

L'auteur visualise les informations du registre de routage comme un graphe, en considérant les systèmes autonomes comme étant les nœuds, et les annonces de routage envoyés et en attente de réception comme étant les arcs entre les nœuds. Après avoir la représentation graphique, l'auteur trouve le plus petit chemin dans le graphe en utilisant l'algorithme de Dijkstra (*o*ne-to-all shortest path algorithm) pour que chaque miroir crée les chemins de SA pour chaque SA vers le SA contenant le miroir le plus « proche ».

La redirection du client vers le miroir le plus « proche » peut se faire en utilisant l'entête

Location du protocole http qui sait rediriger ses clients vers n'importe quel autre URL. Dans ce cas, http prendrait l'adresse IP du client, consulterait la table de « mirorage » pour trouver le miroir le plus « proche » et retournerait 'entête *Location* avec ce miroir.

Cette solution présente deux problèmes majeures : i) la notion de « proche » comme étant le minimum de SAs traversés est fausse et ii) les SAs ne sont pas tous de la même taille (donc cette mesure n'est pas indicative).

2.7.4 AKAMAI

Akamai [Aka] est une société fournissant plusieurs services au dessus d'Internet tels les services de distribution de flux HTTP et vidéo ainsi que des services de monitoring et autres. Ces services sont fournis au dessus d'un réseau mondial lui appartenant éparpillé à travers Internet et comportant quelques milliers de serveurs.

Le service phare, sans doute le plus utilisé, est celui de la distribution du flux HTTP, FreeFlow. Les serveurs d'Akamai sont distribués partout dans le monde et couvrent ainsi une grande partie du réseau mondial, ce qui fait qu'ils sont géographiquement proches des utilisateurs.

La première étape consiste en l'« Akamaisation » du site voulant bénéficier des services. Ceci implique un changement des documents sur le serveur d'origine en modifiant les liens à l'intérieur des pages pour qu'elles pointent sur le réseau Akamai. Lorsqu'un site « Akamisé » reçoit une requête, il répond en envoyant la page modifiée et FreeFlow délivre les objets à l'intérieur de la page du serveur Akamai le « plus adapté », i.e le « plus proche » et le « moins chargé ».

Nous n'avons pas trouvé des publications concernant directement Akamai. Toutefois, des articles ont été publiés [HBLL99, KSB+99] par des membres de la société Akamai qui nous mène à croire que le système utilisé par Akamai est fortement inspiré des techniques décrites par ces articles.

2.7.5 Moteur de Recherche de miroirs

Le travail de [BB99] permet de trouver des miroirs parmi un ensemble de URLs. Leur travail propose une technique qui permet la détection de couples des serveurs mirorés étant donnée un grand ensemble de URLs. L'ensemble n'a pas besoin d'être compréhensif, c'est à dire, l'ensemble total des *paths* dans tous les serveurs n'ont pas besoin d'être présents. La procédure pour la détection des miroirs consiste en deux étapes : la détection des couples candidats et la classification des couples serveurs.

– **Détection des couples candidats**

  Dans cette étape sont considérés les URLs disponibles pour chaque serveur dans la base de données. Un échantillon de ces URLs sont processés pour trouver des similitudes syntaxiques entre ces sous-ensembles. Le résultat de cette étape est une liste de couples serveurs qui sont potentiellement miroirs un de l'autre.

– **Classification des couples serveurs**

  Dans cette étape la liste de candidats de l'étape précédente est alors processée en testant à chaque cas si les serveurs sont des vrais miroirs et en estimant l'extension de leur superposition.

  Ce travail est intéressant surtout parce qu'il permet une découverte des miroirs d'une façon autonome. Il faut tout de même remarquer que c'est un travail immense de comparer les

URLs disponibles sur l'Internet et leur contenu. De ce fait, nous croyons invraisemblable que cette technique puisse être appliqué sur Internet dynamiquement. En outre, le coût en nombre des messages s'avère prohibitif.

2.8 Résumé

Pour faire face aux problèmes dus au large échelle, deux types d'architectures sont intéressantes : l'organisation hiérarchique des entités du système et la coopération entre entités pour l'échange d'information. Ces deux approches ne sont pas mutuellement exclusives et l'utilisation conjointe du modèle hiérarchique et du modèle de coopération peut être une solution efficace.

La réplication de données dans les réseaux étendus peut diminuer la latence d'accès et le temps de réponse au client. Différentes approches sont utilisées en fonction du grain à répliquer, de la durée de vie des réplicats, de leur politique de placement et du protocole de maintien de la cohérence. Le choix de ces paramètres intervient directement sur le mécanisme de désignation approprié et le système de localisation à utiliser.

Les systèmes de désignation existants ne savent pas localiser plusieurs copies d'une même donnée de façon transparente. Actuellement, un objet peut être désigné par son nom de façon symbolique et par son adresse d'une manière physique. Pour atteindre l'objet lui-même, il est nécessaire d'utiliser un mécanisme de localisation. Les services de nommage font le lien entre un nom et une adresse pour chaque espace de nommage.

Dans les réseaux étendus, le nom d'un objet est, en principe, composé de plusieurs espaces de nommages. Cela garantit un nom unique pour chaque objet, laissant à chaque espace de nommage une liberté dans l'attribution du nom de ses objets. Les systèmes de nommage utilisés actuellement dans le large échelle incluent des informations sur la localisation de l'objet dans leur nom. Cela facilite le traitement d'un plus grand nombre de données et améliore le temps de localisation. Cependant, l'objet ne peut pas migrer et des objets identiques placés dans des endroits différents n'auront pas le même nom.

La découverte de données se fait par des mécanismes de recherche par mots-clés appelés *moteurs de recherche*. Les *moteurs de recherche* sont des outils intéressants pour la localisation d'une information en générale mais n'a pas été conçu pour localiser un objet spécifique.

Dans le cas de la réplication des miroirs sur le réseau Internet, quelques solutions ont été proposées pour leur localisation et leur découverte. Les solutions basées sur le système DNS [Moc87] proposent soit de rediriger, de manière statique, vers des miroirs différents selon le serveur de nom choisi, soit de modifier la base de données de DNS pour pouvoir lier un nom à plusieurs adresses. La deuxième solution augmente considérablement le volume de données stockées par chaque serveur de nom, mais elle nous semble intéressante si elle est couplée à un mécanisme de choix de la copie la plus appropriée. La première solution nous semble appropriée aux miroirs les plus populaires et d'une durée de vie longue.

Un réplicat à gros grain et avec une durée de vie « persistante », comme les miroirs, facilite la mise en place des mécanismes de localisation et des protocoles de cohérences. Il est souhaitable dans ce cas qu'une stratégie de placement supportant le partitionnement réseau soit prévu. En outre, un protocole de cohérence optimiste est mieux adapté aux problèmes de latence présents dans un réseau à large échelle.

Dans la suite de cette thèse nous présentons un système de localisation des miroirs $AR\tilde{A}$. $AR\tilde{A}$ propose une architecture coopérative pour localiser des réplicats à gros grain et ayant une durée de vie assez longue sur le Web. $AR\tilde{A}$ travaille avec le nommage URL et avec le système DNS pour permettre aux utilisateurs d'accéder aux différentes copies d'une donnée de façon transparente. L'échange d'information des méta-données[11] entre groupes d'utilisateurs, donne un caractère dynamique à la localisation dans $AR\tilde{A}$. La coopération par groupe permet le passage à l'échelle par son aspect extensible dans la communication épidémique inter-groupe.

11. Données sur la localisation des réplicats

Deuxième partie

$AR\tilde{A}$: spécification et réalisation

Chapitre 3

Conception

Sommaire

3.1 Introduction

L'objectif dans la conception de $AR\tilde{A}$ est de fournir un système capable de localiser, de façon transparente, les miroirs qui intéressent à un certain groupe d'utilisateurs. Le but est d'exploiter les données répliquées dans des miroirs pour diminuer le temps d'accès à un document et gagner en bande passante. Nous voulons savoir où se trouve les miroirs sans pour autant mettre en œuvre un mécanisme global qui localise tous les miroirs mais qui coûte cher en ressources et en temps. La solution consiste à localiser seulement les miroirs susceptibles d'être accédés. Ne connaissant pas à l'avance les miroirs qui seront accédés, nous nous basons sur la localité d'intérêts entre groupes d'utilisateurs pour anticiper un accès éventuel à un miroir. Pour cela, une coopération étroite entre organismes permet d'échanger des informations sur la localisation des miroirs connus de chacun. $AR\tilde{A}$ repose sur un protocole coopératif entre organismes qui permet l'échange épidémique de méta-données (information sur la localisation).

3.2 Caractéristiques de l'environnement

Dans cette section nous décrivons l'environnement de fonctionnement du système $AR\tilde{A}$ pour mieux comprendre certaines choix de conception. Nous expliquons les bases de DNS et nous présentons les éléments de solution de l'application Web. Une brève description des URLs, du langage HTML et du protocole HTTP sont fournis. Ensuite, un résumé des caractéristiques des accès Web sont exposés. Enfin, les propriétés des miroirs sont mis en évidence par rapport aux choix faites dans le système $AR\tilde{A}$.

3.2.1 L'application Web

Le système de localisation des miroirs $AR\tilde{A}$ a été conçu pour fonctionner dans un environnement large échelle tel que le Web. Le Web (World Wide Web) est un système de partage d'informations [SEKN92, DC94] qui utilise normalement l'Internet comme réseau. C'est une application qui offre une interface homme-machine à tel point qu'aujourd'hui le Web se confond avec l'Internet.

Le Web permet l'accès à plusieurs services existants sur Internet. Il donne accès à des documents Hypertext, mais également à des serveurs *ftp*, des serveurs *Gopher*, des serveurs *Wais*, etc. Le Web utilise le réseau de transport Internet pour transporter de l'information et s'appuie sur :

- un système de désignation des noms URL (*Uniform Resource Locator*) ;
- un langage d'écriture de pages d'information HTML (*HyperText Markup Language*) ;
- un protocole de transport HTTP (*HyperText Transfer Protocol*) ;
- des logiciels clients pour la navigation tels que *Netscape* ou *Microsoft Explorer*.

Internet et le système de nommage DNS

L'Internet utilise le DNS (*Domain Name System*) [Moc87] pour désigner les machines dans son réseau. Le système DNS permet un nommage unique de toutes les machines du réseau Internet. Les machines sont organisées en domaines. À chaque domaine correspond un espace de nommage géré par une autorité administrative. Les espaces de nommage sont reliés par un arbre hiérarchique. Les espaces de nommage offrent un système d'identification de machines homogène et unique. C'est à dire, tous les utilisateurs associent le même nom à chacune des machines de l'Internet.

Un des mécanismes essentiel pour le fonctionnement du Web, est le fait que les systèmes connectés peuvent porter un nom unique dans le monde, indépendamment de leur adresse physique. C'est le système de nommage DNS qui permet de transformer ces noms en adresse.

Désignation URL

Le nommage dans le Web est assuré par des URLs. Les URLs permettent un nommage universel des ressources et permettent d'accéder tous les documents mis à disposition par la communauté Internet. Un URL a le format suivant :

nom_du_protocole ://nom_de_la_machine/nom_d'un_fichier_sur_cette_machine

ou

nom_du_protocole ://nom DNS/nom_d'un_fichier_sur_cette_machine

Un exemple d'URL est :

http ://www-sor.inria.fr/publi/RDSWCDC_icdcs99.html

La résolution d'un nom se fait en parcourant trois espaces de nommage : le protocole, le nom de la machine et le nom du document.

Langage HTML

Les objets nommés par des URLs sont des documents texte, des images, des sons, etc. Ces documents peuvent être dynamiques et répondent souvent à une recherche dans une base de données, à une commande ou à un programme (*script*).

Le langage HTML permet la création des documents qui contiennent des liens vers d'autres documents. Ces documents sont représentés par des URLs.

Le langage HTML permet également la création des documents avec texte enrichi, avec images, tableaux, liens hypertexte, images cliquables. Il est aussi possible d'inclure de films et de sons, préparer une interface de saisie, etc.

Protocole HTTP

Le protocole HTTP repose sur un principe simple de la télécommunication. Pour entraîner une action sur un ordinateur distant — telle que demander l'envoi d'une page d'information — l'utilisateur n'a pas besoin d'avoir un compte sur cet ordinateur ni d'établir une connexion pendant la phase de dialogue. Un simple message est suffisant s'il décrit entièrement la fonction à réaliser sur l'ordinateur distant, et indique qui formule cette demande. Ce message peut être véhiculé de manière asynchrone de point à point.

Navigateurs Web

Les navigateurs Web sont des logiciels clients qui permettent d'utiliser les services offerts sur le Web. L'utilisateur se sert d'un navigateur pour émettre sa requête (donner l'URL du document recherché) et pour recevoir les données, les afficher, etc.

3.2.2 Caractéristiques des accès Web

L'étude faite sur les traces des accès Web (voir la section 5.4.1) nous a montrée que le modèle de coopération peut répondre efficacement au problème de localisation des miroirs. L'étude a porté sur la concentration du trafic dans les accès aux serveurs Web et sur la localité d'accès entre organismes.

Concentration du trafic

Parmi toutes les requêtes observées, nous constatons que la plus grande partie de ces requêtes se dirige vers une minorité de serveurs. En outre, nous remarquons que la grande majorité du trafic (en nombre d'octets) provient d'un petit sous-ensemble du total des serveurs.

La concentration du trafic favorise la politique de coopération. Si le nombre de miroirs qui peut améliorer la performance et diminuer le trafic est petit, il n'est pas intéressant de mettre en œuvre un système global pour localiser les miroirs. Le coût inféré par l'échelle de la globalisation des informations est prohibitif si les informations à d'intérêt commun ne sont pas nombreuses.

Localité d'accès entre organismes

Le trafic étudié sur trois *organismes* (voir la section 5.4.2) présente une localité d'accès significative. Les serveurs communs accédés se trouvent parmi les serveurs responsables de la majorité du trafic. Ces résultats encouragent fortement le modèle de la coopération pour la localisation des miroirs. Si on a des miroirs des serveurs les plus accédés et que la localisation de ces miroirs seront échangées via $AR\tilde{A}$, nous pouvons conclure que la localité d'accès entre les *organismes* coopérants indique le taux de coopération utile. Si la localité d'accès entre les différents *organismes* coopérants est grande, l'échange d'informations entre ces *organismes* est souhaitable. Les *organismes* peuvent profiter de la coopération et améliorer ainsi le temps d'accès.

3.2.3 Propriétés des miroirs

Dans cette section nous présentons certaines propriétés observées dans les miroirs. Nous expliquons la représentation de données, l'aspect persistant des entités contenues dans les miroirs, la durée de vie assez longue des miroirs et finalement leur gestion de la cohérence.

Représentation en arborescence

En général, les miroirs proposent la réplication de tout un serveur ou la réplication d'une sous-arborescence d'un serveur. L'intérêt est de répliquer des objets qui présentent une relation logique entre eux voire thématique et qui se trouvent regroupés géographiquement sur un site.

Le modèle d'arborescence obéit à un système de nommage hiérarchique. Cela permet de localiser tous les objets par une seule adresse. L'adresse de la racine de l'arborescence ou du serveur est donc suffisante pour localiser tous les documents (fichiers) qui appartiennent à cette hiérarchie. Ce modèle permet aux miroirs d'être identifiés par un préfixe d'URL représentant toute leur arborescence.

Cette propriété permet au système de localisation de réduire les informations concernant la localisation des documents. La correspondance entre le nom d'un document et sa localisation est de l'ordre du nombre des documents. En utilisant la représentation en arborescence la correspondance revient à stocker l'adresse de la racine de l'arborescence. Pour localiser tous les objets qui appartiennent à l'arborescence une seule entrée est suffisante.

$AR\tilde{A}$ tire profit de cette propriété pour mieux représenter la localisation (voir la section 3.5) des données répliquées (une seule entrée dans la table de localisation par arborescence) et pour diminuer le volume des données de localisation échangées entre les groupes d'*organismes*.

Réplication persistante

Le miroir peut être défini comme une réplication persistante des données où les documents répliqués sont le plus souvent les mêmes. Contrairement aux caches où nous ne pouvons connaître quels documents sont cachés, dans le miroir nous connaissons les documents présents.

La réplication persistante a l'avantage de faciliter le travail de localisation. Vu que la liste des documents repliqués est peu modifiée, une fois connue leur localisation, cette information peut être utilisée plusieurs fois. La conséquence est la réduction du nombre des messages nécessaires pour mettre à jour le système de localisation.

ARÃ tire profit de la propriété de *persistance* pour optimiser l'algorithme de recherche d'un document : si le nom du document obéit à un système de nommage hiérarchique (comme expliqué dans la section précédente) la recherche peut se faire directement sur l'adresse de la racine de son nom.

Durée de vie

Les miroirs présentent une durée de vie assez longue, contrairement à la réplication faite par des caches où les objets sont volatiles et de courte durée. Un miroir peut avoir une durée de vie de l'ordre des mois voire des années pour les serveurs le plus stables. Dans le meilleur des cas, un miroir peut avoir la durée de vie égale à la durée de vie du serveur source *miroré*.

Nous pouvons ainsi faire des choix de communications paresseuses entre les organismes, surmontant ainsi les contraintes du synchronisme dans un environnement large échelle.

Cette propriété favorise aussi la gestion des informations sur la localisation des miroirs. L'information sur l'adresse d'un miroir est stable et le coût des mises à jour pratiquement nul. Seule la vérification de l'existence des miroirs s'avère nécessaire.

ARÃ profit de cette propriété pour mieux gérer la coopération entre les organismes (voir les sections 3.4.1 et 4.5.1) (envoyer les messages une seule fois) et pour optimiser la communication.

Cohérence

Les données d'un miroir ne pouvant être modifiées, les réplicats sont en lecture seule. Les modifications se font uniquement sur le serveur source qui est le seul à avoir la version courante (à jour).

La gestion de la cohérence entre les miroirs est beaucoup plus facile dans ces conditions. Le serveur source centralise la gestion de versions qui motivent la mise à jour d'un miroir. Il est donc aisé de comparer la version d'un miroir par rapport à la version actuelle.

ARÃ utilise cette propriété pour gérer la connaissance des versions des données répliqués. Comme expliqué dans la section 3.5.2, la version d'une donnée est utilisée dans les critères de choix du miroir à accéder.

3.3 Découverte des miroirs dans *ARÃ*

Dans cette section nous décrivons la solution choisie par *ARÃ* pour découvrir des miroirs. Nous expliquons le modèle de coopération utilisé et ses avantages dans un environnement à large échelle. Nous définissons l'entité *organisme* et l'architecture d'une cellule. Enfin, nous montrons le fonctionnement de la coopération à travers la formation des groupes coopérants.

3.3.1 Le modèle de coopération

La coopération est une technique qui permet à des groupes d'entités de partager des informations. Le modèle de coopération est basé sur l'idée simple de la décentralisation. Le modèle de coopération se base sur l'échange des informations par des entités qui présentent des intérêts communs et qui sont proches géographiquement. La coopération est un modèle extensible souvent utilisé dans le large échelle. L'extensibilité est la capacité d'intégrer de nouveaux participants au modèle sans pour autant être obligé de le reformuler ou de le redimensionner. Ce côté facile à gérer favorise l'utilisation de ce modèle dans le large échelle.

Dans l'Internet, le modèle de coopération est une solution pour diminuer le coût d'accès à l'information et le temps de communication. Si l'information recherchée peut être trouvée sur un site plus proche, on gagne en temps d'accès et en trafic réseau. Or, la coopération évite des accès redondants par le groupe des entités qui coopèrent.

La collaboration entre plusieurs entités évite les goulots d'étranglements et tous les problèmes dus à la centralisation : surcharge, panne, etc. De même, une trop grande répartition peut entraîner des surcoût prohibitif des protocoles. Il faut trouver un compromis : la coopération inter-*organismes*.

Un système de localisation transparente des miroirs basé sur la coopération est un mécanisme intéressant pour des *organismes* qui partagent des intérêts communs. $AR\tilde{A}$ est un système de localisation de miroirs basé sur la coopération entre *organismes* et groupes d'*organismes* (voir la figure 3.3). $AR\tilde{A}$ découvre les miroirs par échange d'informations sur leur localisation avec d'autres *organismes*. Pour cela, le choix des *organismes* doit être judicieux. C'est la localité entre ces *organismes* qui assure l'efficacité de $AR\tilde{A}$.

Des études faites [BB99] montrent que la quantité des miroirs accédés est petite. Cela peut s'expliquer par le fait que les miroirs sont encore en expansion, mais aussi parce que les intérêts des utilisateurs sont concentrés sur quelques sites. L'accès ciblé des utilisateurs justifie la localisation des miroirs par coopération avec d'autres utilisateurs. Nous pensons qu'une solution par coopération peut être suffisante pour découvrir les miroirs qui intéressent un certain groupe d'*organismes* (voir la section 3.3.2). La coopération dans $AR\tilde{A}$ se justifie par les caractéristiques des traces d'accès Web. Dans ces traces nous pouvons observer la localité entre *organismes* (voir la section 5.4.2) sur les accès Web. Cela montre que les *organismes* étudiés présentent des intérêts communs. Le modèle de coopération est donc une solution intéressante de ce point de vue. Le trafic est redirigé vers les serveurs miroirs connus grâce à la coopération avec les autres *organismes*.

3.3.2 Organisme et groupes d'organismes

Un *organisme* (voir la figure 3.1) est une entité formée par plusieurs sites géographiquement proches et liés par des intérêts communs. Nous appelons cet ensemble des sites un *organisme* $AR\tilde{A}$.

Ces *organismes* sont formés de façon explicite par les administrateurs systèmes et s'identifient dans le système de coopération. Un *organisme* peut représenter une entreprise, un institut ou un campus universitaire ou un laboratoire de recherche. C'est à ce niveau que la localité d'accès va être présent avec un taux d'accès commun profitable et un coût de coopération acceptable. Si le grain de l'organisme est plus petit, par exemple à niveau des projets d'un institut, la quantité de méta-données de localisation récolté par les accès du groupe n'est pas significative et donc pas profitable au gain en performance. En plus, le coût de la coopération inter- *organismes* croît, étant donnée que le nombre des groupes augmente. Si au contraire, nous augmentons le grain

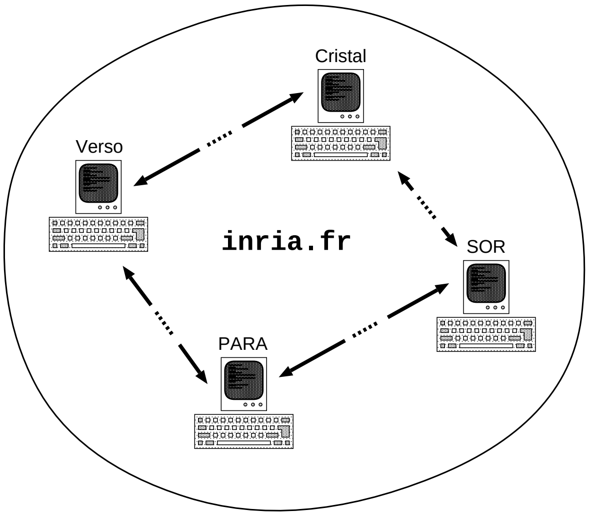

FIG. 3.1 – *Un organisme*

de la formation d'un groupe, le serveur $AR\tilde{A}$ aura un coût d'accès plus grand qu'un accès sur un réseau local; le coût de l'utilisation de $AR\tilde{A}$ ne doit pas égaliser le gain obtenu lors de la localisation des miroirs.

Architecture $AR\tilde{A}$ d'une *cellule*

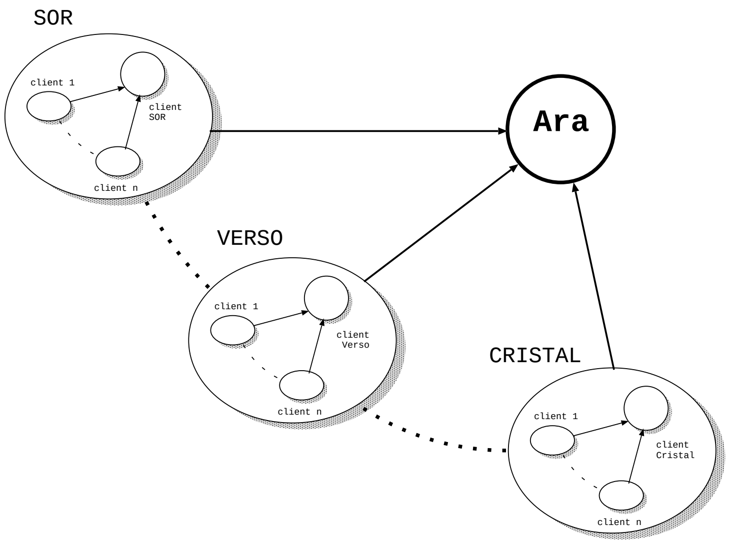

FIG. 3.2 – *Architecture $AR\tilde{A}$ d'une cellule*

Un *organisme* $AR\tilde{A}$ actif est denominé *cellule*. Cellule est donc un *organisme* qui utilise le système $AR\tilde{A}$ pour localiser des miroirs et que participe aux échanges des messages avec d'autres *organismes* pour connaître et faire connaître la localisation des miroirs.

Nous pouvons observer l'architecture d'1 *cellule* $AR\tilde{A}$ dans la figure 3.2. Chaque *cellule* présente un niveau d'hiérarchie que facilite la communication avec le serveur $AR\tilde{A}$ local. Le niveau d'hiérarchie correspond à des projets dans un laboratoire de recherche, à des facultés dans un campus universitaire, à des départements dans une entreprise, etc.

Coopération inter-organismes

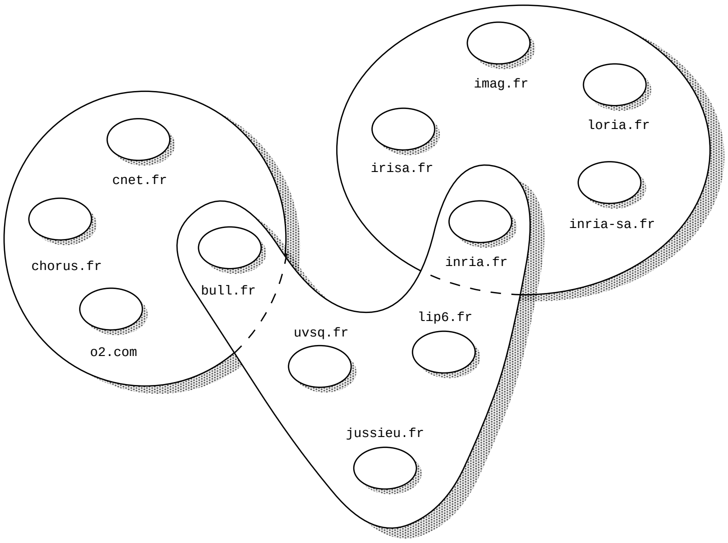

FIG. 3.3 – *Groupes d'organismes*

Les *organismes* ainsi formés peuvent coopérer. Les *organismes* ayant des intérêts en commun et qui appartiennent à une même région géographique sont de bons candidats à une coopération fructueuse. Les *organismes* partageant des intérêts communs ont de fortes chances d'avoir une localité significative dans la recherche des documents sur le Web. De même, les *organismes* bénéficiant d'une bonne liaison réseau auront un coût d'échange d'informations plus bas.

Un *groupe d'organismes* (voir la figure 3.3) forme une entité de coopération où l'échange d'information est active. Le groupe d'*organismes* est une *cellule* clé dans la conception de $AR\tilde{A}$. C'est dans le groupe d'*organismes* que la découverte des miroirs se fait. Chaque *organisme* participant apporte sa coopération en partageant avec le groupe la localisation des miroirs qu'il connaît.

Un *organisme* peut coopérer avec plusieurs *groupes d'organismes*, faisant partie de plusieurs *cellules* de découverte de miroir. Interagir avec plusieurs *cellules* permet de connaître un plus grand nombre de miroirs. Cependant, le coût de la communication entre groupes augmente.

Formation des groupes

La formation des groupes passe par deux étapes : définition (abonnement) du groupe et connexion réseau.

- **Définition**
  La définition des *organismes* participants du groupe ou la politique d'abonnement des *organismes* participants du groupe.
- **Connexion**
  Le maillage complet entre les participants du groupe ou le graphe connexe entre les participants du groupe.

On distingue deux types des groupes : des groupes ouverts et des groupes fermés. Les groupes ouverts se font connaître d'un grand nombre d'*organismes* et acceptent leur abonnement selon la proximité géographique. Les groupes fermés sont connus que par les *organismes* définis par les administrateurs systèmes et n'acceptent pas des abonnements avant une entente préalable.

Groupes ouverts

Les groupes ouverts sont crées par un *organisme* et acceptent des abonnements d'autres *organismes* selon des critères (voir la section 4.5.2) propre à chaque groupe. Les critères d'autorisation d'adhésion sont basés sur la distance géographique et leur domaine d'intérêts. À la longue, les *organismes* parasites de l'échange de l'information peuvent être exclus du groupe.

- **Définition**
  Dans un premier temps un *organisme* crée le groupe avec lui seul comme participant. Ensuite, il informe aux administrateurs systèmes des autres *organismes* que le groupe existe. Les *organismes* voulant coopérer au sein de ce groupe déclenchent la procédure d'abonnement.

- **Connexion**
  Les groupes ouverts forment un graphe connexe où un *organisme* ne connaît pas nécessairement tous les membres du groupe. L'échange des messages dans le groupe se fait par épidémie selon que l'*organisme* récepteur du message n'ait pas encore reçu le message. Si l'*organisme* récepteur a déjà reçu le message il ne donne pas suite à la diffusion du même. Pour contrôler l'arrivée des messages, chaque *organisme* stocke le numéro du message, l'*organisme* émetteur et son groupe

- **Collaboration individuelle**
  Le créateur d'un groupe ouvert peut être un fournisseur de service. Il fournit comme service les informations sur la localisation des miroirs aux individus qui n'appartiennent pas à un *organisme* au sens *ARÃ* ; c'est la *collaboration individuelle*. Un individu peut s'abonner à un fournisseur de service dans une politique de groupe ouvert pour former un *organisme virtuel* et coopérer comme entité unique avec des *organismes ARÃ*.

Groupes fermés

Les groupes fermés visent une coopération très ciblé et donc moins chère et plus efficace. Dans un groupe fermé le nombre des membres est réduit minimisant ainsi le nombre de messages. L'aspect plus ciblé dans la coopération suppose un centre d'intérêt commun entre les participants du groupe. Dans ce cas, les messages échangés ont une forte chance de véhiculer de données utiles aux autres membres du groupe. En plus, la restriction de la taille du groupe permet à chaque *organisme* d'avoir une vision globale du maillage du groupe.

- **Définition**
  Dans les groupes fermés, les participants des groupes sont définis par les administrateurs système selon une entente préalable. Tout *organisme* voulant adhérer à un groupe doit connaître le responsable de ce groupe et doit obtenir une autorisation d'abonnement auprès de ce dernier.
- **Connexion**
  Les connexions dans les groupes fermés sont point à point et sont établies au moment de l'abonnement au groupe. L'*organisme* qui s'abonne à un groupe fermé doit connaître tous les autres membres du groupe pour mieux gérer les échanges des messages. Les messages sont envoyés par l'*organisme* émetteur vers tous les membres du groupe, diminuant ainsi les nombre des messages échangés à l'intérieur du groupe.

3.3.3 Stratégies de découvertes de miroirs

La découverte des miroirs peut se faire de deux façons différentes : interactive et coopérative.

Interactive

Dans ce cas, l'administrateur système informe le système $AR\tilde{A}$ de la localisation des miroirs. L'administrateur peut inscrire les données référentes à un miroir et sa source à travers l'interface $AR\tilde{A}$ (voir la section 4.7.1). L'ajout d'un miroir ou la création d'une entrée pour un nouveau serveur source se fait par le même procédé.

Coopérative

Une coopération inter-*organismes* (voir la section 3.4.1) permet au système $AR\tilde{A}$ d'échanger des informations sur les miroirs afin de découvrir de nouveaux miroirs et de mettre à jour les informations sur les miroirs déjà catalogués. La coopération peut être déclenchée par la publication ou le partage.

- **Publication**
  Le créateur d'un miroir peut rendre public la localisation de son miroir. Pour cela, l'administrateur fait connaître le miroir localement à travers une manipulation interactive de l'interface offerte par $AR\tilde{A}$. Ensuite, $AR\tilde{A}$ constate l'existence d'un nouveau miroir, inconnu des autres *organismes* avec qui il coopère, et déclenche la procédure de partage d'informations.
- **Partage**
  Le module local de $AR\tilde{A}$ reçoit d'un autre *organisme* un message avec l'information de la localisation d'un miroir inconnu. $AR\tilde{A}$ inclut ce nouveau miroir dans sa base de données et effectue une vérification pour savoir quels sont les *organismes*, avec qui il coopère, qui n'ont probablement pas reçu cette information. Ensuite, $AR\tilde{A}$ envoie un message à chacun de ces *organismes*.

3.4 Protocole épidémique

Le protocole épidémique consiste à envoyer des messages d'un point à un autre point. Le point récepteur envoie lui aussi le message à d'autres points, diffusant ainsi une information par contamination « *voisin-voisin* ». Le protocole épidémique assure donc la coopération à travers une communication point à point et asynchrone. Les protocoles épidémiques sont adaptés au large échelle par leur aspect extensible qui les permet d'assimiler de nouveaux participants sans coût administratif supplémentaire. Le protocole épidémique gère aussi le filtrage des messages et les cycles. Par la suite nous expliquons comment la coopération a lieu dans le protocole épidémique. Ensuite, nous détaillons la façon par laquelle la communication se fait et comment le protocole gère le grand nombre de messages.

3.4.1 Coopération inter-organismes

La coopération se fait entre groupes d'*organismes*. Un *organisme* voulant coopérer doit s'abonner auprès d'un groupe. L'*organisme* s'engage à partager les informations de localisation des miroirs avec le groupe. En échange, l'*organisme* reçoit des informations sur la localisation d'autres miroirs, provenant des autres membres du groupe.

La coopération inter-*organismes* se traduit par un protocole d'échange d'informations. Un *organisme* échange des informations après avoir été informé lui même de l'existence d'un miroir.

Tous les *organismes* sont avertis de l'existence d'un miroir dès qu'un des *organismes* du groupe :

- découvre et répertorie un miroir ;
- crée et repértorie un miroir ;
- est informé par $AR\tilde{A}$ de l'existence d'un miroir et le repértorie ;

L'envoi de la localisation d'un miroir à un *organisme* se fait après vérification que le message n'est pas redondant (i.e., qu'il s'agit d'un nouveau miroir) et selon une gestion de dates (voir la section 3.7.3). Une *cellule* $AR\tilde{A}$ vérifie que les *organismes* avec lesquels existe une coopération font partie du même groupe d'*organismes* que l'émetteur du message. Dans ce cas, $AR\tilde{A}$ n'envoie pas des messages à ces *organismes* ; le message sera envoyé à ces *organismes* directement par l'émetteur.

La coopération inter-*organismes* a un coût correspondant au nombre des messages envoyés vers et reçus d'autres *organismes*. $AR\tilde{A}$ minimise ce coût à travers des stratégies de communication.

Coopération multiple

Chaque *organisme* peut s'abonner à plusieurs groupes. La coopération avec plusieurs groupes fournit un accès plus riche aux informations. D'une part, cela permet de coopérer avec des groupes qui s'intéressent à différents sujets ; d'autre part, cela augmente le nombre d'informations échangées sur la localisation des miroirs. Cependant, la coopération multiple aura un coût en messages supplémentaires. Désormais les messages sont envoyés à tous les *organismes* de tous les groupes auxquels l'*organisme* émetteur appartient (voir la section 3.4.2). En plus, l'intersection entre groupes peut créer des cycles. Nous discutons sur la gestion des cycles dans la section 3.4.3.

3.4.2 Communication dans $AR\tilde{A}$

$AR\tilde{A}$ utilise la communication de groupe pour assurer la coopération entre *organismes*. Chaque nouveau message est diffusé vers tous les groupes auxquels l'*organisme* participe. La gestion de la communication est faite dans chaque groupe d'*organismes* et les connexions entre les *organismes* sont point à point.

Cohérence entre les méta-données

La communication asynchrone est possible dans $AR\tilde{A}$ parce que la cohérence entre les méta-données (sur la localisation des miroirs) n'est pas nécessaire. Si un *organisme* connaît un miroir et un autre *organisme* du même groupe ne le connaît pas, ce dernier pourra toujours accéder aux données à la source ou sur un autre miroir. De plus, en cas de panne, la non coopération est transparente pour le reste du système.

L'échange sur la localisation des miroirs peut améliorer les performances à la longue. À terme, tous les *organismes* auront la même vision des miroirs, l'effet immédiat n'étant pas le plus important.

Échange épidémique des messages

Le protocole $AR\tilde{A}$ utilise la communication épidémique point à point [PSTT95]. Ce choix se justifie dans la mesure où ce moyen de communication est tolérant aux fautes, ne nécessite pas la connaissance de tous les participants et facilite le traitement des déconnexions.

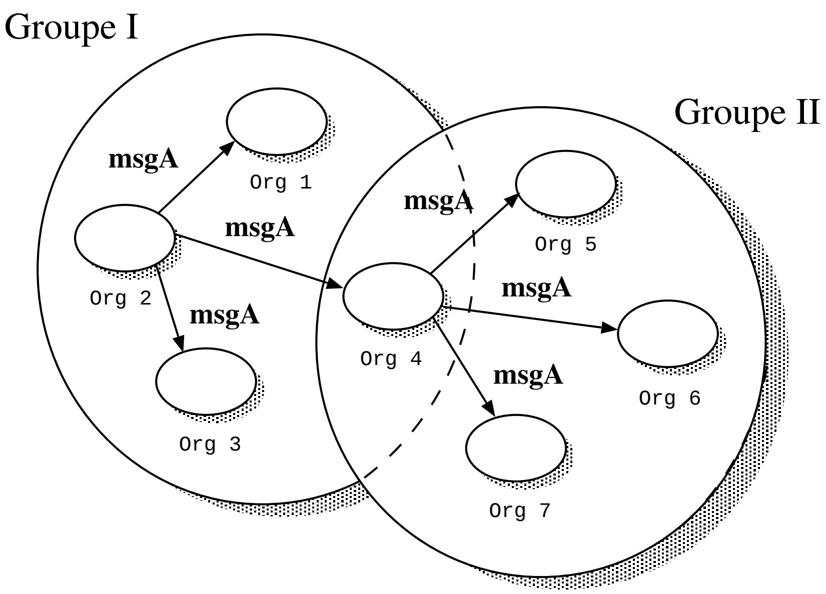

FIG. 3.4 – *Échange épidémique*

L'échange d'information par épidémie est un modèle qui passe à l'échelle et profite de la topologie actuelle de l'Internet. Cependant, cette méthode peut créer de cycles exigeant un contrôle sur les messages déjà reçus et envoyés (voir la section 3.4.3), pour ne pas faire transiter des messages inutiles.

Dans la figure 3.4, nous illustrons un échange épidémique interne au *groupe I* issu de l'*organisme org2* et un échange épidémique externe au *groupe I* engendré par l'*org4* vers le *groupe II*. L'*org2* émet un message à tous les membres du *groupe I* à travers des communications point à point avec chaque membre du groupe. Chaque membre reçoit le message directement de l'émetteur (*org2*) par épidémie. L'échange externe au groupe a lieu parce que l'*org4* pratique une coopération multiple avec le *groupe I* et le *groupe II*. Quand l' *org4* reçoit le message provenant de l'*org2* il transmet ce message à tous les membres du *groupe II* par épidémie de degré 2 (voir la section 3.4.3).

Communication asynchrone

L'information sur la localisation des miroirs est échangée d'une façon paresseuse. Le temps de vie d'un miroir est assez long pour justifier une propagation paresseuse de sa localisation. L'avantage d'une propagation paresseuse est le gain en bande passante.

L'échange des messages dans $AR\tilde{A}$ est asynchrone. L'échange d'informations se fait d'une façon opportuniste. Les messages $AR\tilde{A}$ sont traités lorsque le serveur n'est pas trop chargé. La diffusion épidémique est ensuite déclenché d'une façon paresseuse. $AR\tilde{A}$ envoie les messages groupés et différés (*batch*) ou à l'expiration d'un temporisateur et éventuellement en *piggybacking*.

Un des avantages de la communication asynchrone est de faire travailler le serveur $AR\tilde{A}$ quand il est oisif et non quand il est trop chargé. Une autre avantage est la réduction du nombre de messages sur le réseau. La communication dans $AR\tilde{A}$ économise donc la bande passante et améliore le temps de réponse au client.

3.4.3 Filtrage de l'information

Le partage de l'information par coopération a l'avantage de sélectionner naturellement les informations selon le groupe avec lequel la coopération a lieu. Cependant, dans un modèle de coopération où il existe des intersections entre les groupes, certaines informations ne sont pas nécessairement intéressantes. Dans ce qui suit nous discutons les problèmes liés à l'intersection entre les groupes et le filtrage nécessaire de l'information.

Degré de contamination

Une façon de filtrer les informations à échanger avec les autres groupes est donc d'établir un *degré de contamination*. Nous entendons par *degré de contamination* une métrique pour savoir jusqu'à quel niveau d'indirection sur la coopération, l'information reste admissible.

Le *degré de contamination* est un compteur que s'incrémente chaque fois que le message traverse une frontière entre deux groupes d'*organismes*. Le *degré de contamination* est un indice qui aide chaque *organisme* à décider si il accepte ou non un message venant d'un *organisme* qui appartient à un groupe différent du sien. Si le message a traversé plusieurs *groupes d'organismes*, le *degré de contamination* sera grand. Plus le *degré de contamination* est grand, plus petite la probabilité de que le message soit intéressant.

Chaque *groupe d'organismes* décide du *degré de contamination* qui lui convient. Chaque *organisme* sera donc contaminé uniquement par les messages d'une proximité calculé. La proximité est mesurée par le nombre d'organismes qui le message traverse. Le *degré de contamination* doit refléter une certaine probabilité de localité entre les *groupes d'organismes* voisins. Le choix du *degré de contamination* peut être changé par le *groupe d'organismes* à tout moment.

Cette solution a l'avantage d'être simple. De plus, si les groupes de coopération ont une tendance naturelle à chercher la localité, cette solution doit en plus fournir de bons résultats.

Condition d'épidémie

Nous définissons `DgCM` (`Degré de Contamination d'un Message`) comme étant le *degré de contamination* correspondant à un message à un moment donnée de son parcours. Chaque message transporte son `DgCM` qui représente le nombre des *groupes d'organismes* traversés.

Nous définissons `DgCGi` (`Degré de Contamination du Groupe i`) comme étant le *degré de contamination* fixé par le groupe i. Tous les messages acceptés par le groupe i doivent respecter la limite imposée par le *degré de contamination* choisi.

Nous appelons la limite imposée par le *degré de contamination* la ***condition d'épidémie***. La ***condition d'épidémie*** impose que le `DgCM` soit inférieur ou égal au `DgCGi`. Cela garantie que les messages en circulation dans un groupe i proviennent d'un groupe éloigné au maximum de `DgCGi` groupes.

Condition d'épidémie :

```
DgCM <= DgCGi
```

Dans la figure 3.5 nous pouvons observer l'évolution d'un message ainsi que son degré de contamination. L'*org2* envoie un message à son groupe (Groupe 1) et son degré de contamination est affecté à 1. L'*org 4*, qui fait partie du Groupe 1, reçoit le message. L'*org 4*, qui fait aussi partie du Groupe 2, par épidémie envoie ce message aux membres du Groupe 2. Cela est possible parce que le degré de contamination du Groupe 2 est égal à 3. Cela signifie que le Groupe 2 accepte de recevoir de messages qui sont passés par trois groupes différents avant. Or, le message en question est passé par un seul groupe et présente donc un degré égal à 1.

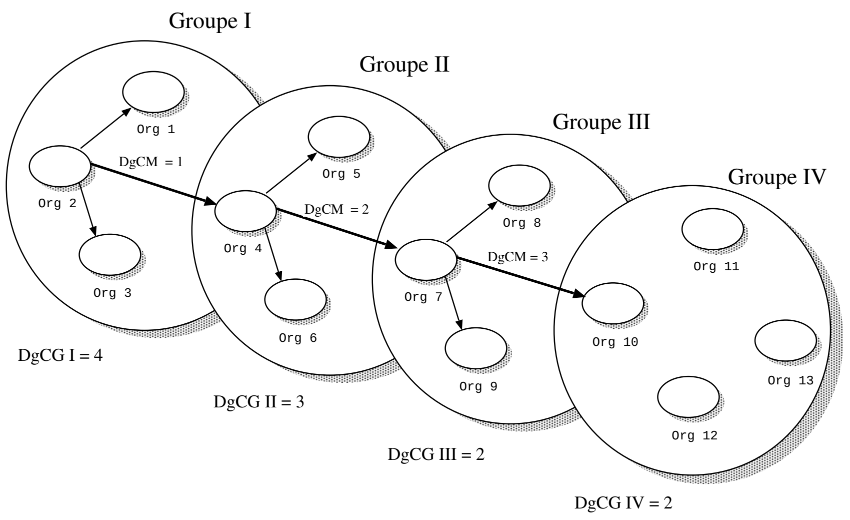

FIG. 3.5 – *Degré de contamination dans les messages*

```
DgCM = 1 (Degré de contamination du message = 1)

DgCG2 = 3 (Degré de contamination du groupe 2 = 3)

DgCM <DgCG2
```

Cela obéit la ***condition d'épidémie*** :

```
DgCM <= DgCGi
```

L'*org 7* qui appartient au Groupe 2 et au Groupe 3, reçoit le message avec un degré de contamination égal à 2 et par épidémie l'envoie au Groupe 3. Le Groupe 3 a son degré de contamination réglé à 2 et peut donc accepter le message.

L'*org 10* reçoit le message provenant de l'*org 7* avec un degré de contamination de 3. L'*org 10* décide de ne pas propager le message au Groupe 4 parce que le degré de contamination de ce dernier est égal à 2 et ne satisfait pas la condition d'épidémie (`DgCM` $>$ `DgCG4`).

Les cycles des messages

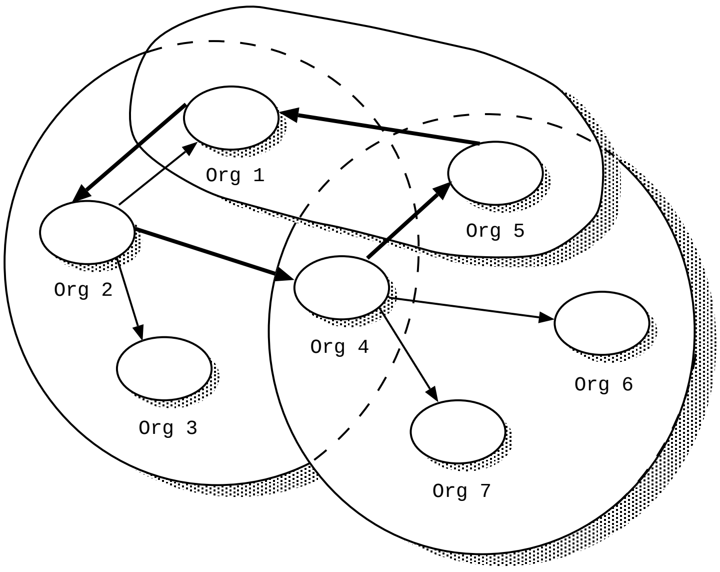

FIG. 3.6 – *L'existence des cycles entre groupes d'organismes*

Supposons plusieurs groupes qui coopèrent entre eux. Un groupe coopère avec un autre groupe à travers un de ses membres. Un *organisme* peut coopérer avec plusieurs groupes. Cela forme des graphes connexes entre les *organismes* appartenant à des groupes différents. La formation des cycles inter-groupes dans le graphe peut créer des cycles de messages. Les cycles peuvent générer des messages redondants et ainsi dégrader la performance du protocole et du réseau.

La figure 3.6 montre ce scénario. Nous pouvons observer à travers les flèches en gras la création d'un cycle. *Org 2* envoie un message vers son groupe. *Org 4* reçoit le message et l'envoie vers un autre groupe. *Org 5* reçoit à son tour le message et l'envoie vers un autre groupe avec qui il coopère. *Org 1* reçoit alors le message provenant d'*Org 5*. Cependant, *Org 1* avait déjà reçu le même message en provenance d'*Org 2*. *Org 1* peut alors envoyer le message à *Org 2* si un contrôle n'est pas effectué. Dans les deux cas, des messages inutiles ont été envoyés : le message de l'*org 5* vers l'*org 1* et le message de ce dernier vers l'*org 2*.

Une façon d'éviter les cycles des messages est le *degré de contamination* qui rejette les messages qui ont traversé un certain nombre de *groupes d'organismes* (selon le choix de chaque *organisme*). Cependant, la technique du *degré de contamination* ne peut pas garantir totalement la non formation des cycles puisque chaque groupe est indépendant dans le choix de son *degré de contamination*. Cela peut créer de zones de libre circulation des messages où l'intersection entre les groupes permet la création des cycles. Pour cela, une technique de *ramassage de miettes* (voir section 3.7.1) est mis en place.

3.5 Localisation des miroirs

Localiser un miroir peut être difficile et l'utilisateur peut préférer accéder aux données directement à la source, exigeant plus de bande passante et sacrifiant une meilleure performance. Le support de localisation transparente des miroirs vise à éliminer ce problème. Un utilisateur pourra accéder au miroir le plus « approprié » en ayant connaissance de l'URL source ou d'un de ses réplicats. Un mécanisme d'accès transparent aux données repliquées permet à l'utilisateur d'accéder à n'importe quelle copie de la donnée sans qu'il en soit conscient.

3.5.1 Solution pour le nommage

Un système reparti doit permettre de donner des noms uniques aux entités qui le composent. Sur le Web les noms sont uniques selon leurs adresses IP, c'est pourquoi une même donnée placée en deux endroits différents n'a pas le même nom. $AR\tilde{A}$ propose une solution pour le nommage des réplicats qui utilise une table de correspondance entre le nom de la source et le nom du réplicat.

Dans l'Internet le nom d'un document est lié à son adresse IP. Nous avons d'abord l'adresse du domaine (du serveur) et ensuite le chemin de la sous-arborescence du système de fichier. Un miroir peut avoir donc le même nom que la source à niveau du suffixe (arborescence du système de fichier) et un préfixe différent de celui de la source (adresse IP de la machine ou adresse du domaine).

Pour résoudre le problème du nommage, la correspondance se fait entre le préfixe des URLs des serveurs source et les préfixes des URLs des serveurs miroir. Comme montre l'exemple ci-dessous : le serveur source du MIT a un miroir à l'INRIA et un autre à Paris VI.

$AR\tilde{A}$ identifie par comparaison de préfixe, l'entrée correspondante à l'URL de la requête. Ensuite, $AR\tilde{A}$ choisit le miroir à accéder parmi les miroirs catalogués. $AR\tilde{A}$ sait aussi identifier (voir les sections 3.5.1 et 4.7.1) un URL, émis par la requête, parmi la liste des miroirs. Dans ce cas, l'utilisateur fournit un URL correspondant à un miroir et $AR\tilde{A}$ propose un autre miroir plus proche ou identifie son serveur source.

```
http ://www.mit.edu --> http ://www.inria.fr/mit.edu/
                        http ://www.jussieu.fr/mit.edu/
                                          ⋮
```

Sous-miroirs

Il existe des miroirs qui répliquent seulement une partie d'un serveur. Nous les appelons des sous-miroirs. Les sous-miroirs ont un préfixe plus long pour représenter la racine de leur arborescence. Par exemple, supposons un sous-miroir du MIT qui réplique seulement la partie sur les systèmes informatiques. Nous pouvons avoir deux entrées dans la table capables de répondre à une même requête de données. Comme montre l'exemple ci-dessous : le serveur source du MIT a un miroir complet à l'INRIA et autre à Paris VI et un sous-miroir à l'INRIA au projet SOR et un autre à Paris VI au LIP6. Quand nous cherchons dans la table de correspondance un miroir, l'idéal est de trouver le miroir qui correspond au préfixe le plus long de la source.

```
http ://www.mit.edu -->         http ://www.inria.fr/mit.edu/
                                http ://www.jussieu.fr/mit.edu/
http ://www.mit.edu/systems --> http ://www-sor.inria.fr/mit.edu/systems/
                                http ://www.jussieu.fr/lip6/mit.edu/systems/
                                                  ⋮
```

Recherche du préfixe

L'URL cherché par une requête est comparé avec les URLs des serveurs sources de la table de correspondance. Ensuite, si la correspondance n'est pas trouvée, l'URL est recherché parmi les URLs des miroirs. $AR\tilde{A}$ prévoit l'existence d'autres miroirs qui peuvent être accédés plus rapidement ou qui possèdent une version plus récente de la donnée recherchée. Enfin, pour mieux prendre en compte le problème de sous-miroirs, nous recherchons la correspondance la plus longue à niveau du préfixe.

Structure de données

La structure de donnée choisie pour stocker les préfixes est un arbre binaire léxicographique équilibré. Une recherche dans une telle structure permet d'atteindre le niveau d'identification le plus long à niveau du préfixe si souhaité. En plus, l'équilibre de l'arbre garantit un parcours plus rapide. La hauteur de l'arbre dans ce cas étant de l'ordre de $log_2(n)$; où n est le nombre des miroirs connus par $AR\tilde{A}$.

L'utilisation d'un arbre binaire léxicographique se justifie dans la mesure où le système $AR\tilde{A}$ présente beaucoup d'accès en lecture, peu de modifications sur le contenu de l'arbre et donc peu de rééquilibrage de l'arbre. Le grand nombre d'accès en lecture est dû aux requêtes des clients pour la localisation des miroirs. Les modifications de l'arbre consiste à ajouter et supprimer des miroirs. L'ajout des miroirs est supposé avoir une période de pic au début de l'utilisation de $AR\tilde{A}$ ou au moment d'un abonnement à un groupe ; à part ça, les ajouts des miroirs ne sont pas si nombreux au point de créer un coût significatif dans le rééquilibrage de l'arbre. La suppression des miroirs est dans le plupart des cas liée à la durée de vie du miroir. Les miroirs ont une durée de vie assez longue et ne génèrent donc pas beaucoup des suppressions dans les arbres.

3.5.2 Critères de choix d'un miroir

Choisir le *meilleur* miroir est un problème important dans le modèle de découverte des miroirs [MDZ98]. C'est le choix du miroir qui permet de satisfaire les besoins de l'utilisateur. Accéder à un mauvais miroir peut coûter cher en temps et en ressources.

Le *meilleur* miroir obéit à différents critères qui requiert une mise à jour constante. Tout cela a un coût à comparer aux avantages de connaître le *meilleur* miroir au moment de la requête.

Différents critères peuvent être retenus pour choisir le miroir. Le choix va dépendre de la priorité des besoins de chaque *organisme* ou de chaque utilisateur. Nous énumérons quelques critères qui nous semblent pertinents dans le cadre du problème étudié :

Temps d'accès

Le temps d'accès vers un miroir vise à diminuer le temps de réponse de l'utilisateur et à économiser la bande passante. Le temps d'accès vers un miroir permet de savoir quels sont les miroirs le plus « proches ». La « distance » [Tac97] peut être mesurée par le temps d'aller retour d'un paquet sur le réseau ou par le nombre des routeurs traversés pour atteindre le miroir. Le temps d'accès peut être mesuré (voir le chapitre 5) et permet de choisir le miroir qui offre l'accès le plus rapide.

Charge du serveur

Pour diminuer le temps de réponse, le temps d'accès réseau n'est pas le seul critère déterminant. Le temps de traitement de la requête par le serveur est aussi important. La charge du serveur est un indice pour estimer le temps du traitement de la requête. Le serveur le moins chargé offre normalement l'accès le plus rapide.

La charge du serveur peut être mesurée par le nombre des clients dans une période de temps donnée. La puissance de la machine est aussi un critère à prendre en compte pour calculer la charge d'un serveur. Cependant, côté client, la charge d'un serveur est aperçue par le temps du traitement d'une requête, soit le temps de réponse infligé à l'utilisateur qui a émis la requête.

Dans $AR\tilde{A}$, la charge est mesuré par des calculs faits à la volé par le proxy client. Le proxy effectue des calculs statistiques basé sur la variation du temps de réponse d'un client par rapport à un serveur donnée. Le proxy client informe $AR\tilde{A}$ de toute dégradation importante de temps de réponse observé pendant les accès de son client. Cela implique une coopération entre le serveur $AR\tilde{A}$ et le proxy client.

Version

La version d'un miroir est un critère important quand la donnée recherchée a été modifiée récemment, subit des modifications régulières ou vient d'être publiée. Supposons un utilisateur que cherche à avoir un article publié récemment. Dans ce cas, le besoin de l'utilisateur est d'accéder la donnée la plus récente et non pas à celle dont l'accès est le plus rapide.

3.5.3 Interaction avec le cache

Afin de profiter de tous les réplicats présents dans notre système, il faudrait prendre en compte les réplicats qui se trouvent dans les caches aussi.

Les différences majeures entre caches et miroirs peuvent se résumer en deux points :

1. les caches ont une durée de vie limitée, soit par besoin de cohérence ou par besoin d'aménagement d'espace.
2. les caches ont une politique de réplication dirigée par les requêtes de leurs clients, alors que les miroirs sont plutôt des entités de réplication guidées par les changements sur les serveurs eux mêmes.

De ce fait, il paraît plus logique que les caches soient plus proches des clients que les miroirs (sauf pour les cas spéciaux des miroirs montés dans l'*organisme* lui-même).

Ainsi, nous aboutissons à une suggestion de système final de localisation qui englobe la coopération entre caches et miroirs (voir figure 3.7).

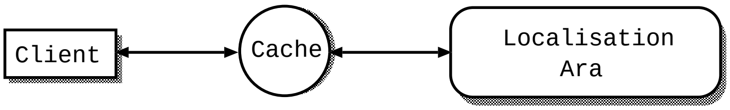

FIG. 3.7 – *L'interaction avec le cache*

Pour assurer l'accès le plus rapide la requête du client va chercher d'abord la donnée dans le cache. Si la donnée se trouve dans le cache, l'accès $AR\tilde{A}$ n'a pas lieu. Dans le cas contraire, la requête est redirigée vers $AR\tilde{A}$. Si $AR\tilde{A}$ connaît des miroirs pour la requête correspondante, $AR\tilde{A}$ répond par une liste avec tous les miroirs connus de sa base de données. Le système va ensuite rechercher dans le cache si un de ces documents est présent. Si oui, l'accès est local (dans le cache), sinon, l'accès sera fait vers le miroir choisi.

Si $AR\tilde{A}$ reconnaît dans la requête l'existence d'un miroir local, il va chercher la donnée directement sur le miroir local. Dans ce cas, la recherche dans le cache n'est pas intéressante.

3.6 Cohérence incrémentale

La *cohérence incrémentale* permet aux utilisateurs de voir un objet mis à jour de façon incrémentale par rapport au temps. La *cohérence incrémentale* garantit que les opérations de lecture d'un objet (d'une donnée) contient toutes les modifications déjà observées auparavant par cet utilisateur.

3.6.1 Scénarios de perte de cohérence

Le problème de la perte de cohérence peut survenir lorsque le serveur d'accès change pendant la consultation du client. Il y a plusieurs raisons pour lesquelles un changement de serveur d'accès a lieu. Les principales raisons sont : (i) la déconnexion ou le partitionnement réseau, (ii) l'interaction avec le cache.

- **Déconnexion ou partitionnement réseau**
  Lorsqu'un serveur (miroir) n'est plus accessible dû à une déconnexion de ce dernier ou à un partitionnement réseau, le système $AR\tilde{A}$ va choisir un autre réplicat pour remplacer le serveur inaccessible. Si le réplicat choisi a une version moins récente que celle du serveur précédent, l'utilisateur peut observer des incohérences des données.

- **Interaction avec le cache**
  Dans le cas où le document recherché se trouve dans le cache, ce document peut avoir une version moins récente que celle consultée par l'utilisateur. Ce dernier peut donc observer des incohérences dans les données pendant la consultation.

3.6.2 Non retour sur le passé

Le problème du *retour sur le passé* se pose lorsqu'un miroir devient inaccessible ou lorsqu'il y a une interaction avec le cache. L'utilisateur qui est en train d'accéder à ce miroir aura une erreur et cherchera à relancer la requête. Le problème surviendra si le miroir alors connecté possède une version plus **ancienne** que celle que l'utilisateur avait déjà consultée.

ARÃ assure à ses utilisateurs le *non retour sur le passé* pour chaque document consulté. Cela veut dire que l'utilisateur peut se trouver avec une copie différente de celle qu'il a consulté auparavant, mais cette nouvelle copie sera toujours plus récente que la précédente.

ARÃ offre le *non retour sur le passé* de façon transparente. L'utilisateur ne voit pas l'effet de la déconnexion ni la Redirection de la requête.

Le *non retour sur le passé* peut être assuré de deux façons :

- **Chercher directement à la source**
  Dès que le problème survient, l'utilisateur est aiguillé directement à la source. Cette dernière ayant sûrement la version la plus récente.

- **Connaître la dernière version de tous les miroirs accédés par chaque utilisateur**
  La connaissance de toutes les versions accédées par chaque utilisateur, permet le choix d'un autre miroir qui a une version équivalente ou plus récente que celle accédée auparavant.
  Cette solution requiert le stockage de la dernière version de tous les miroirs accédés par chaque utilisateur. Cela peut causer des problèmes de place en mémoire ou en disque. D'un autre côté, le *meilleur* miroir peut être choisi lors d'une déconnexion imprévue. Cela implique un éventuel gain en latence et en bande passante. Ces informations peuvent aussi être utilisées pour la mise à jours des miroirs.

Assurer le *non retour sur le passé* est garantir une *cohérence incrémentale* à la vue de l'utilisateur. Dans *ARÃ*, la *cohérence incrémentale* est assurée par des *sessions de cohérence* comme expliqué dans la section suivante.

3.6.3 Sessions de cohérence

Pour assurer la *cohérence incrémentale* dans la vue d'un utilisateur, *ARÃ* offre la *session de cohérence.* Une *session de cohérence* assure à une suite des requêtes une cohérence incrémentale sur les données téléchargées. Une *session de cohérence* est une abstraction pour la séquence de lecture et d'écriture effectué pendant l'accès d'un utilisateur à cet objet. Elle permet à l'utilisateur d'avoir une vue cohérente avec un *non retour sur le passé.* Une *session de cohérence* n'assure pas une cohérence entre les données, à la place elle propose une vue individuelle et cohérente avec les actions d'un seul utilisateur.

La garantie de la *cohérence incrémentale* impose une restriction sur le sous-ensemble des serveurs (miroirs) qui peuvent être utilisés pendant une session. Demander une garantie de cohérence peut avoir un impact inverse sur la disponibilité et sur le temps de réponse. Les utilisateurs doivent faire un compromis entre cohérence et disponibilité, entre cohérence et meilleur temps de réponse.

L'avantage d'assurer la cohérence par le biais des sessions réside dans le fait que seul l'utilisateur intéressé par une cohérence aura moins des serveurs disponibles et sera affecté par un temps de réponse plus important. Les autres utilisateurs de *ARÃ*, même ceux de la même *cellule*, ne payerons pas le prix d'une cohérence inutile.

Dans *ARÃ*, la solution choisie pour assurer le *non retour sur le passé* est celle de connaître la version de tous les miroirs accédés par l'utilisateur à l'intérieur d'une *session de cohérence.*

Ouverture d'une session

L'utilisateur voulant travailler sur une vue cohérente doit ouvrir une *session de cohérence.* L'ouverture d'une session se fait en signalant le navigateur Web avant la première requête. Par la suite $AR\tilde{A}$ garantit que les dates de mise à jour d'un document accédé sera croissant par rapport au temps.

Fermeture d'une session

La fermeture d'une *session de cohérence* peut se faire via le navigateur Web à tout moment durant la connexion avec $AR\tilde{A}$. $AR\tilde{A}$ reçoit alors le message de fin de session et rétablit le processus normal de localisation des miroirs. À partir de ce moment, le critère de choix d'un miroir est celui choisi par l'utilisateur (ou administrateur système) comme le plus approprié à ses besoins (par exemple le miroir le plus rapide).

3.7 Contrôle et performance

3.7.1 Ramasse Miettes

$AR\tilde{A}$ propose un *ramasse miettes* des messages en attribuant à chaque message un TTL (Time To Live). Un TTL est représenté par un entier et correspond au nombre d'*organismes* par lesquels le message a le droit de passer. Un TTL est attribué à un message au moment de sa création. Ensuite, à chaque *organisme* traversé, le TTL est décrémenté de un. Lorsque le TTL atteint la valeur zéro, le message n'est plus diffusé.

Cette technique très connu dans l'environnement réseau a l'avantage d'être simple et efficace. Le *ramasse miettes* de $AR\tilde{A}$ est très facile à mettre en œuvre et évite pas l'existence de messages « éternels ». En plus, le *ramassage de miettes* limite le nombre de messages sur le réseau, diminuant ainsi le coût de la coopération.

3.7.2 Validation des miroirs

$AR\tilde{A}$ a une politique de publication et de divulgation décentralisée et donc susceptible d'erreurs et de malveillance. La divulgation d'un miroir inexistant ou la disparition d'un miroir peuvent encombrer le réseau avec des messages inutiles. En plus, l'utilisateur verrait le temps de réponse croître en essayant de contacter des serveurs introuvables.

Afin d'éviter la divulgation des miroirs fantômes, $AR\tilde{A}$ vérifie l'existence des miroirs régulièrement. Si un miroir n'existe plus, sa localisation est supprimée de la base de données. La validation des miroirs se fait par des envois des messages *ping*. Ces messages servent aussi à estimer le temps d'accès vers les miroirs (voir la section 3.5.2).

Quand un miroir est publié, $AR\tilde{A}$ valide son existence pour la première fois. Ensuite, $AR\tilde{A}$ vérifie éventuellement que le miroir existe toujours. Chaque *organisme* peut choisir de ne pas utiliser la procédure de validation, si la connaissance d'un miroir provient d'un membre du groupe fiable par exemple.

3.7.3 Gestion de dates

Dans $AR\tilde{A}$, les messages portent des informations sur l'existence d'un miroir, la fin d'un miroir ainsi que la version d'un miroir. Ces informations (méta-données) peuvent être plus ou moins récentes. La gestion de dates sert à vérifier la cohérence du message par rapport aux informations stockées dans la base de données.

$AR\tilde{A}$ prévoit une gestion des dates afin de maintenir une cohérence incrémentale sur l'information obtenue par les méta-données. Les méta-données avant d'être stockée doivent passer par une vérification de comparaison de leur date avec la date des méta-données déjà connues.

Une information sur la date de mise à jour d'un miroir va être comparée avec la date de mise à jour connue de ce même miroir. Si la date (version) connue de ce miroir est plus ancienne que la date communiquée, alors la date sera mise à jour dans la base de données de miroirs (BDM). Si, au contraire, la date connue est plus récente que celle communiquée, $AR\tilde{A}$ n'effectue pas de modification dans la BDM.

Dans le cas d'une suppression d'un miroir, $AR\tilde{A}$ vérifie si la date de ce miroir est plus récente que la date du même miroir dans la base de données. Si la date dans la BDM est plus récente, $AR\tilde{A}$ ne peut pas être sûr que le miroir n'existe plus réellement. Dans ce cas, $AR\tilde{A}$ garde la localisation de ce miroir.

3.8 Résumé

$AR\tilde{A}$ est un système pour la découverte des miroirs basé sur le modèle de la coopération. Nous avons utilisé les propriétés spécifiques aux miroirs pour proposer une solution adaptée. En plus, une étude sur les caractéristiques des accès Web par rapport aux miroirs a été menée pour mieux comprendre les besoins réels des utilisateurs. Basé sur cette étude le modèle de coopération a été choisi.

$AR\tilde{A}$ a été conçu pour le Web et utilise donc les URLs comme système de nommage. $AR\tilde{A}$ identifie par comparaison de préfixe les URLs correspondants aux mêmes données (miroirs). Le système DNS est utilisé, dans la requête final vers le serveur ou le miroir, à fin de localiser physiquement le serveur à accéder.

Le système $AR\tilde{A}$ a pour entité de base les *cellules* composées d'un *organisme* et d'un serveur $AR\tilde{A}$. Un ensemble des cellules forment un *groupe d'organismes* qui coopèrent entre eux pour la découverte et la localisation des miroirs. La coopération entre *groupes d' organismes* est aussi possible grâce au protocole d'échange épidémique des messages qui assure le passage à l'échelle.

Le protocole épidémique d'échange de messages est basé sur une communication point à point et sur un *degré de contamination* qui contrôle le filtrage des informations entre les différents *groupes d'organismes*.

$AR\tilde{A}$ se divise en deux parties : le support de localisation transparente des miroirs et le protocole de coopération inter- *organismes* pour la découverte des miroirs. La première partie sert à rendre transparent, vu de l'utilisateur, la localisation des miroirs ; la deuxième partie permet de connaître, avec un surcoût minime, la localisation des miroirs connues de chaque *organisme* grâce à l'utilisation d'un protocole paresseux. Le système de localisation $AR\tilde{A}$ vise à améliorer les performances de temps d'accès et peut augmenter la disponibilité de données.

Chapitre 4

Réalisation

Sommaire

4.1 Introduction

Dans ce chapitre, nous présentons l'architecture $AR\tilde{A}$ mise en œuvre dans le prototype développé en Java. Cette architecture comporte un protocole de découverte des miroirs et un système de localisation des miroirs. Nous présentons tout d'abord l'environnement de développement, le déploiement du système $AR\tilde{A}$ et son modèle orienté objet. Nous montrons, dans un deuxième temps, l'implémentation du protocole pour la découverte des miroirs et du système de localisation des miroirs. Nous décrivons ensuite les structures de données utilisées et nous concluons.

4.2 L'environnement

Dans cette section nous décrivons brièvement l'environnement système utilisé dans le développement de $AR\tilde{A}$ et le support réseau sur lequel repose sa conception et sa réalisation.

4.2.1 Système

Le prototype $AR\tilde{A}$ a été développé au-dessus d'Unix en utilisant une machine virtuelle Java. Tous les composants de $AR\tilde{A}$ sont des classes Java. Ce choix se justifie par la portabilité offerte par Java et le système Unix, et la facilité de développement dans le langage Java.

$AR\tilde{A}$ est un système composé de plusieurs éléments logiciels coopérants distribués sur plusieurs sites. Comme tout système réparti, $AR\tilde{A}$ utilise une couche de communication pour assurer le fonctionnement en réparti du système. $AR\tilde{A}$ emploie un mécanisme de communication point à point et échange des messages par épidémie entre les sites. Cela favorise l'aspect extensible d'un système coopérant et facilite le passage à l'échelle.

$AR\tilde{A}$ localise des miroirs dans le cadre de l'application Web. L'application Web est le système d'accès aux services sur Internet. Le nommage dans le Web est assuré par des URLs. Les URLs permettent un nommage universel des ressources et permettent d'accéder tous les documents mis à disposition par la communauté Internet.

L'Internet utilise le DNS (*Domain Name System*) [Moc87] pour désigner les machines dans son réseau. $AR\tilde{A}$ se sert de DNS pour localiser les miroirs tout en ayant son propre mécanisme de gestion de aliases (plusieurs noms pour un seul objet).

4.2.2 Environnement réseau

$AR\tilde{A}$ est implanté sur plusieurs sites sur Internet. Internet est un réseau étendue qui présente certaines limitations, en particulier il est impossible de réserver de la bande passante et donc nous ne pouvons pas garantir une qualité de service. De plus, Internet est un réseau hétérogène offrant différents débits.

De façon à pouvoir mettre au point et tester les stratégies de localisation, un prototype $AR\tilde{A}$ a été développé sur un réseau local à 100 Mbits/s. Les tests effectués ont pris en compte l'environnement Internet (voir le Chapitre 6) dans le calcul du temps d'accès. La coopération mise en place a été faite sur des sites différents communicants à travers le Web.

Chaque composant de $AR\tilde{A}$ réside dans un intranet. Nous considérons, pour chaque composant, un système composé d'un ensemble de stations de travail (ou *nœuds*), inter-connectées par un réseau local. Les composants $AR\tilde{A}$ communiquent entre elles en utilisant les sockets Java et sont appelés *cellules*.

$AR\tilde{A}$ traite les documents qui utilisent le protocole HTTP sur le Web. Le protocole HTTP a besoin d'un simple message qui décrit entièrement la fonction à réaliser sur l'ordinateur distant, et indique qui formule cette demande pour entraîner une action sur un ordinateur distant.

4.3 Déploiement $AR\tilde{A}$

Le système $AR\tilde{A}$ est déployé en cellules pour chaque organisme d'un groupe coopérant. Une cellule est associée à un organisme possédant un seul module de $AR\tilde{A}$ même si il coopère au sein de plusieurs groupes. Les stations de travail appartenant à une cellule, utilisent $AR\tilde{A}$ pour accéder au Web.

4.3.1 Services de *ARÃ*

La mise en œuvre de *ARÃ* s'appuie sur trois services : un service de coopération de groupe, un service de résolution du nommage et un service de support transparent d'accès aux différents miroirs.

Dans la figure 4.1 nous pouvons comprendre comment est mis en oœuvre la localisation transparente des miroirs :

- L'utilisateur fait une requête normale via un navigateur Web ;
- Le *proxy* dirige la requête vers le serveur *ARÃ* ;
- *ARÃ* interroge la Base de Données de Miroirs (BDM) sur un URL donné ;
- *ARÃ* répond au *proxy* avec une liste des miroirs ou NULL dans le cas où aucun miroir est connu.

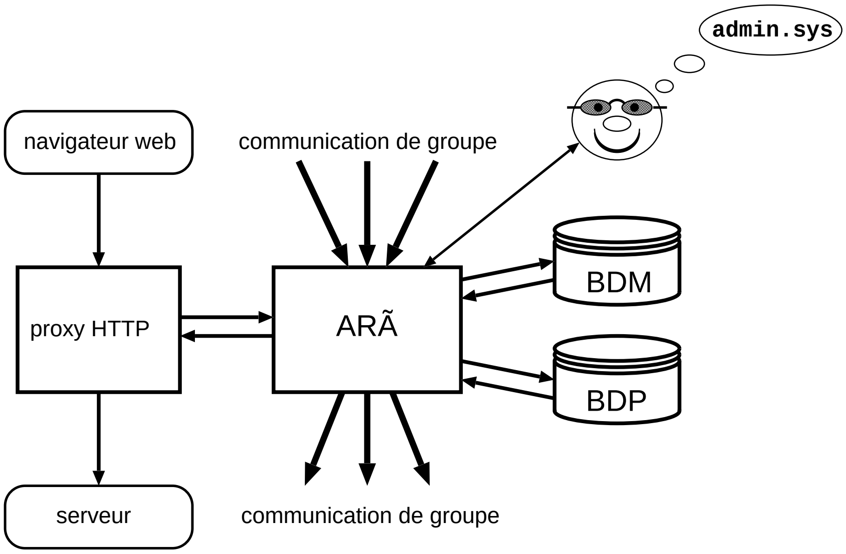

FIG. 4.1 – *L'architecture* ARÃ

ARÃ est capable de résoudre le problème du nommage en utilisant les données acquises via la coopération (voir les sections **??**). L'URL fourni par l'utilisateur peut être l'URL du document source ou l'URL d'une copie de ce document. Dans les deux cas, *ARÃ* est capable d'établir la correspondance et associer l'URL donné à l'URL de la copie la plus appropriée.

La coopération de groupe nourrit la BDM. C'est elle qui découvre les miroirs et les stocke. La coopération fonctionne (voir figure 4.1) de la manière suivante :

- Les messages sur la localisation des miroirs arrivent, provenant des organismes appartenant aux groupes coopérants.
- La cellule *ARÃ* met à jour sa Base de Données de Miroirs (BDM) ;
- Le serveur *ARÃ* interroge la Base de Données de Participants (BDP) pour connaître les organismes vers qui les messages seront propagés ;
- Le serveur de coopération envoie les messages aux organismes coopérants.

4.3.2 Architecture répartie de $AR\tilde{A}$

$AR\tilde{A}$ est composé de plusieurs cellules placées dans des sites différents. Chaque cellule correspond à un organisme. La figure 4.2 montre les composants de chaque organisme :

- un serveur responsable pour la gestion de groupe (*Serveur de Groupe*) ;
- un serveur qui gère la coopération (*Serveur de Coopération*) ;
- un serveur qui assure la localisation transparente (*Serveur http*) ;
- un *proxy* qui s'occupe de la communication avec le client (*Proxy Groupe*) ;
- une Base de Données des Miroirs (BDM) ;
- une Base de Données des Participants (BDP) ;
- une bibliothèque d'accès aux BDs à laquelle sont liées les différents serveurs.

$AR\tilde{A}$ est basé sur l'échange des messages. Il utilise des communications point à point pour la coopération. Le système coopère d'une façon répartie. Le *Serveur de Groupe* reçoit les demandes d'abonnements au groupe et les traite. Le *Serveur de Coopération* reçoit les messages d'information sur la localisation des miroirs provenant des organismes qui appartiennent aux groupes coopérants. Le *Serveur http* traite toutes les requêtes venant du client. Le *Proxy Groupe* envoie les messages d'information sur la localisation des miroirs aux organismes qui appartiennent aux groupes coopérants. Enfin, la BDM (base de données des miroirs – voir la section 4.5) stocke la table de correspondance entre les serveurs sources et ses miroirs et la BDP (base de données des participants) stocke les groupes avec qui l'organisme coopère et ses membres.

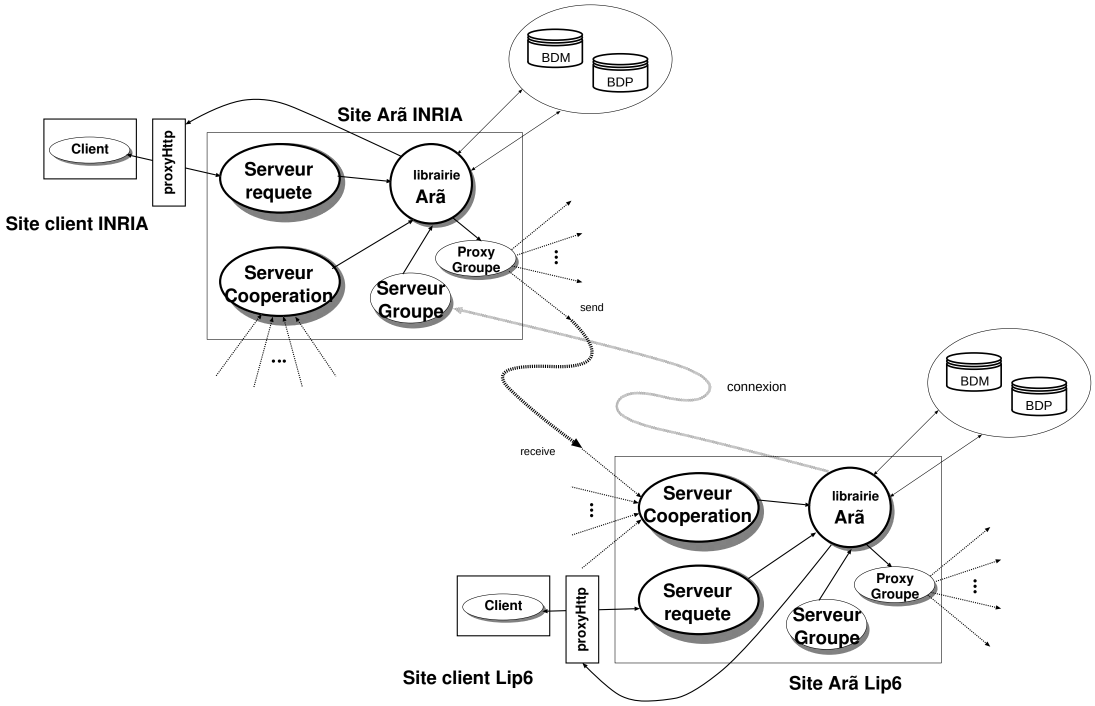

FIG. 4.2 – *Architecture système de* ARÃ

Serveur de Groupe

Le *serveur de groupe* est à l'écoute de demandes de connexions sur un port *ARÃ*. Les demandes de connexion sont une demande de coopération et d'abonnement à un groupe. Le message de demande de connexion contient l'identificateur du groupe auquel l'organisme distant veut s'abonner, ainsi que son identification.

Serveur de Coopération

Le *serveur de coopération* gère l'arrivée des messages contenant des informations sur la localisation des miroirs. Les messages arrivent sur un port d'écoute qui distribue le traitement parmi plusieurs processus assurant ainsi un traitement parallèle. Chacun de ces processus du *serveur de coopération* utilise le *proxy groupe* pour déclencher l'échange épidémique des messages (voir la section 3.4).

Serveur http

Le *serveur http* sert d'interface entre le client et *ARÃ*. Il utilise les primitives de *ARÃ* pour interroger la BDM et pouvoir répondre aux requêtes des clients.

Proxy Groupe

Le *proxy groupe* est le canal de sortie pour toute communication d'une cellule *ARÃ*. Le *proxy groupe* assure tous les envois des messages sur la localisation des miroirs ainsi que les messages sur l'arrivée d'un nouveau membre du groupe ou son départ. Il implante le protocole paresseux épidémique (voir la section 3.4). C'est donc à travers le *proxy groupe* que l'échange épidémique des messages a lieu.

Base de Données des Miroirs (BDM) et Base de Données des Participants (BDP)

La BDM est l'entité de stockage des adresses des miroirs connus de la cellule *ARÃ*. La BDP est l'entité de stockage des organismes appartenant aux groupes avec qui la cellule coopère. La BDM et la BDP peuvent être distribuées entre plusieurs machines ou être centralisées sur une seule machine. Si la BDM ou la BDP est centralisée elle peut se trouver sur la même machine que le serveur *ARÃ* ou sur une autre machine. Dans le prototype actuelle, les BDs sont centralisées mais peuvent se trouver dans une machine différente de celle du serveur *ARÃ*.

4.3.3 Le modèle Client/Serveur

Les cellules *ARÃ* interagissent par échanges de messages. L'interaction entre deux cellules *ARÃ* est du type **client/serveur**. Une cellule *ARÃ* dénommée **client** demande les services d'une cellule *ARÃ* appelée **serveur**. La cellule *ARÃ* cliente est à l'initiative de la demande, c'est elle qui envoie une requête. La cellule *ARÃ* serveur est en attente de requête. Le serveur reçoit la demande du client, traite la demande et répond au client. Une cellule *ARÃ* est à la fois client et serveur.

ARÃ utilise également le modèle **client/serveur** pour réaliser la coopération entre organismes et aussi pour mettre en œuvre le support local d'accès transparent. Dans la coopération entre les organismes, le serveur est l'organisme qui diffuse le message avec la localisation d'un miroir. Les clients sont les organismes qui reçoivent l'information sur les miroirs. Dans le support d'accès transparent le système *ARÃ* sert de serveur pour les navigateurs Web (les clients).

4.4 Le modèle objet

Le système $AR\tilde{A}$ est construit sur le modèle orienté objet. Le formalisme objet facilite la réalisation des systèmes tant il est naturel de modéliser leurs différents composants et leurs interactions en termes d'objets. C'est encore plus vrai avec un système distribué.

Dans $AR\tilde{A}$ tous les modules ARÃ sont basés sur l'utilisation des objets. La communication entre les organismes se fait à travers des objets et la gestion locale de la BDM est totalement faite par des objets. Le serveur $AR\tilde{A}$ et le proxy $AR\tilde{A}$ sont eux-mêmes des objets.

Tout dans $AR\tilde{A}$ est donc un objet. Le système $AR\tilde{A}$ présente donc une modularité à tous les niveaux. Il est donc possible de dynamiser les choix dans $AR\tilde{A}$ d'une façon flexible.

4.4.1 Le choix du langage

$AR\tilde{A}$ a été développé entièrement en Java. Le choix du langage Java obéit à certains critères : (i) Java est un langage totalement orienté objet ; (ii) Java est un langage portable et supporte plusieurs architectures ; (iii) Java est un langage distribué et dynamique ; (iv) Java offre une sécurité dans un milieu distribué ; (v) Java est simple, le temps de développement est réduit.

Cependant, $AR\tilde{A}$ souffre des problèmes de performance rencontrés dans Java. Dans le cadre de cette thèse, la performance peut être absorbée par le fait qu'au niveau du protocole de coopération nous nous intéressons plutôt au nombre des messages échangés et la latence des échanges de messages qu'au temps processeur utilisé. Dans le cas du support de localisation transparent, nous nous intéressons au taux de succès.

4.4.2 Caractéristiques $AR\tilde{A}$ objet

$AR\tilde{A}$ tire profit des propriétés objet et propose une dynamicité et une flexibilité dans son architecture. Pour cela, nous avons voulu limiter au maximum les dépendances entre la partie cliente et la partie serveur dans $AR\tilde{A}$. Du serveur, le client n'a besoin de connaître que le nom et l'interface. L'interface peut donc être implémentée différemment dans chaque cellule ; que ce soit pour les critères de choix du miroir, pour la technique de filtrage de l'information ou encore pour le maintien de la cohérence.

$AR\tilde{A}$ peut offrir une flexibilité pour les critères de choix des miroirs. Pour cela il suffit de définir les critères comme étant des objets et les instancier selon la demande de chaque cellule.

Dans $AR\tilde{A}$ la formation des groupes est dynamique. Les groupes sont des objets qui offrent une interface d'insertion et retrait (`put / remove`) des organismes. De la même façon, l'ensemble des groupes sont catalogués dans un objet qui lui aussi offre une interface d'insertion et retrait des groupes. Les critères d'autorisation pour l'ajout d'un membre dans le groupe sont choisis d'une façon flexible et adaptable selon la politique de chaque groupe.

$AR\tilde{A}$ permet d'ajouter des nouvelles options facilement. En particulier, un module pour la gestion des mises à jour des versions des miroirs est envisageable. La mise à jour des miroirs peut être faite non pas directement à la source mais à travers un autre miroir. En utilisant les données de la BDM de $AR\tilde{A}$ sur les versions des miroirs ainsi que leur « distance », ce nouveau module peut aiguiller la requête d'une nouvelle version vers un des réplicats (miroirs). Ce module n'a été ni spécifié ni implémenté ; mais il constitue une possibilité d'adaptation du système $AR\tilde{A}$ grâce à sa modularité objet.

4.4.3 Modélisation

Dans la figure 4.3 nous montrons la modélisation objet de $AR\tilde{A}$. Chaque cercle représente une classe Java. La classe `ARÃ` est responsable de la création de toutes les entités de communication (`OrganismServer` / `HttpOrganismServer`). `OrganismServer` est en écoute sur le port $AR\tilde{A}$ pour la coopération avec d'autres organismes. `HttpOrganismServer` gère les requêtes des utilisateurs provenant du navigateur Web.

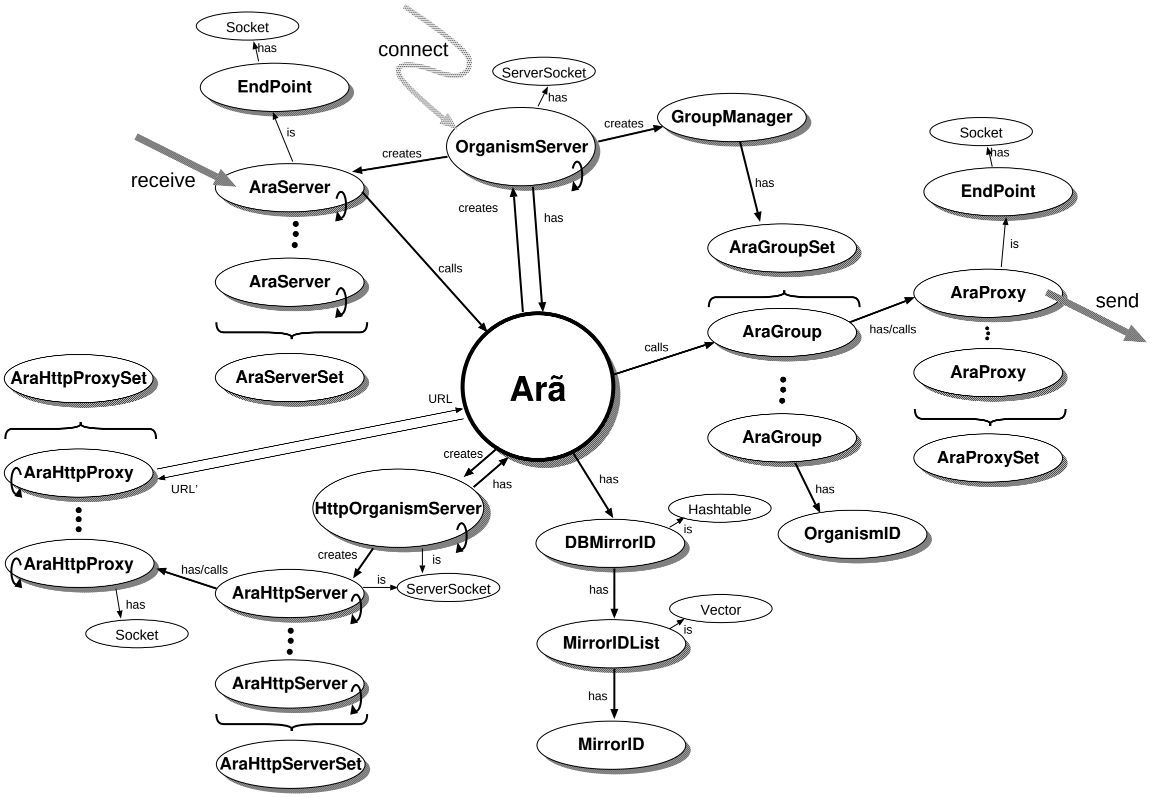

FIG. 4.3 – ARÃ *: modélisation orienté objet*

Organisme et groupes d'organismes

Les organismes et les groupes d'organismes sont représentés par les classes `OrganismID` et `AraGroup`. Dans chaque instance de la classe `OrganismID`, nous avons le nom du site $AR\tilde{A}$ de l'organisme, son adresse IP et son port IP.

Chaque `AraGroup` contient la liste des organismes appartenant à chaque groupe. `AraGroup` contient aussi l'interface de coopération (échange des informations sur les miroirs) ainsi que la gestion de groupe.

L'ensemble des groupes avec qui une cellule $AR\tilde{A}$ coopère est connu de l'instance de la classe `GroupManager` à travers l'instance de la classe `AraGroupSet`. `AraGroupSet` connaît la liste de tous les groupes avec qui la cellule coopère. Le `GroupManager` est responsable de la gestion de groupe en utilisant l'interface `put / remove`. Le `GroupManager` effectue aussi la vérification qui décide selon certains critères pour l'acceptation de l'abonnement (voir la section 4.5.2).

Miroir et groupe des miroirs

Les miroirs et les groupes des miroirs sont représentés par les classes `MirrorID`, `MirrorIDList` et `DBMirrorID`. Dans chaque instance de la classe `MirrorID` nous avons l'URL du serveur source de ce miroir, l'URL du miroir lui même et la date de sa dernière mise à jour.

L'objet `DBMirrorID` contient la liste de tous les serveurs sources connus de cette cellule. Chaque entrée d'un serveur pointe vers un objet `MirrorIDList` qui contient la liste des miroirs correspondant à ce serveur.

Chaque cellule $AR\tilde{A}$ a une instance de la classe `DBMirrorID` où est stockée tous les miroirs connus de cette cellule.

Proxy Http $AR\tilde{A}$

Le **proxy http $AR\tilde{A}$** fait la liaison entre la requête provenant du navigateur (donc de l'utilisateur) Web, les services offerts par $AR\tilde{A}$, le système DNS et enfin le serveur (ou le cache) contenant les données demandées par l'utilisateur.

L'objet `HttpOrganismServer` est un serveur en écoute d'une demande de connexion de la part d'un navigateur Web. Un navigateur Web voulant utiliser $AR\tilde{A}$, demande une connexion à `HttpOrganismServer` qui lui va créer un `AraHttpServer` pour ce navigateur Web. La classe `AraHttpServerSet` connaît l'ensemble de `AraHttpServer` existant.

La classe `AraHttpServer` instanciée est un serveur responsable pour gérer les requêtes provenant du navigateur Web. Chaque requête sera dirigée vers un `AraHttpProxy`.

L'objet `AraHttpProxy` est responsable de faire la liaison entre le client (navigateur Web) et $AR\tilde{A}$. C'est lui qui va établir une connexion avec le serveur $AR\tilde{A}$.

Proxy $AR\tilde{A}$

Le **proxy $AR\tilde{A}$** assure la communication de groupe. Il envoie les messages vers tous les membres des groupes avec lesquels la cellule coopère.

L'objet `AraProxy` (`EndPoint`) assure la connexion vers un autre organisme participant d'un groupe. Pour chaque `AraGroup` un ensemble de `AraProxy` assure les liaisons point à point entre cette cellule $AR\tilde{A}$ et les autres membres de ce groupe. Cet ensemble est catalogué par l'objet `AraProxySet`.

Serveur d'organisme

Le **serveur d'organisme** est responsable de gérer les abonnements à un groupe. Pour chaque cellule $AR\tilde{A}$, et pour chaque *groupe ouvert* de cette cellule, le **serveur d'organisme** reçoit les demandes d'abonnements et les traite jusqu'à l'enregistrement du nouveau membre ou le refus à son abonnement.

L'objet `OrganismServer` est le serveur en écoute de connexion provenant des autres organismes $AR\tilde{A}$. Un organisme voulant coopérer demande une connexion à `OrganismServer` qui vérifie auprès de l'objet `GroupManager` l'autorisation pour cette demande. Pour chaque autorisation donnée, `OrganismServer` crée un `AraServer` responsable de recevoir les messages provenant d'un organisme en particulier. L'ensemble des `AraServers` (`EndPoint`) est catalogué par l'objet `AraServerSet`.

Serveur $AR\tilde{A}$

Le serveur $AR\tilde{A}$ est l'objet `Arã` lui même. Il est responsable de la création de tous les composants de communication de la cellule. C'est l'objet `Arã` qui instancie les objets : `OrganismServer` et `HttpOrganismServer`. L'objet `Arã` est aussi responsable de la gestion des messages à envoyer. Pour cela il fait appel à `AraGroup` qui à son tour fait appel à `AraProxy`.

L'objet `Arã` possède la base de données de miroirs. Dans ce sens, c'est lui qui utilise l'interface d'accès à la BDM. Il faut donc faire appel à l'objet `Arã` pour manipuler la BDM.

4.5 Découverte des miroirs

La découverte des miroirs basée sur l'échange épidémique des messages fonctionne grâce à deux modules : (i) le module de communication entre les organismes et (ii) le module de gestion de groupe.

Le protocole de coopération inter-groupe permet à chaque organisme de publier et échanger les miroirs dont il a connaissance, et en contrepartie de découvrir les miroirs publiés par les autres organismes partenaires. Pour cela, plusieurs fonctionnalités sont fournies à travers un ensemble de messages qui forment le protocole lui même. En plus des messages, une gestion de groupe est nécessaire pour rendre possible une coopération dynamique et efficace (voir la section 4.5.2). Dans ce sens, nous avons défini une base de données, locale à chaque organisme, contenant les organismes coopérants pour chaque cellule. Cette base de données est mise à jour de façon individuelle et permet une indépendance pour chaque organisme par rapport à son groupe de coopération. La figure 4.4 montre la coopération dans un *groupe fermé* (voir la section 3.3.2) et fixe (supposons aucun changement). Par la suite, la question de la formation des groupes ainsi que la possibilité de participer à des groupes différents sera discutée et détaillée.

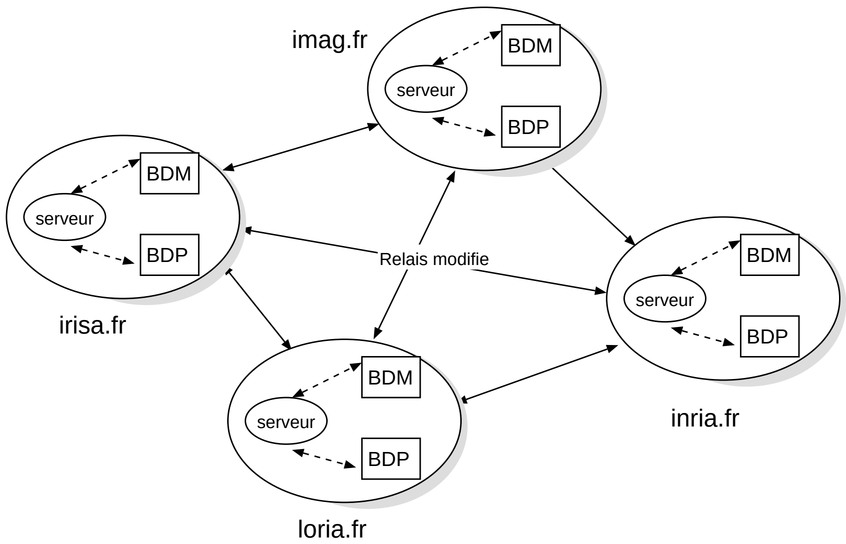

FIG. 4.4 – *Coopération inter-organismes*

4.5.1 Protocole de coopération

Le système de localisation des miroirs $AR\tilde{A}$ que nous proposons repose sur une variation du protocole Relais [MPKD98] pour gérer la communication entre les participants d'un groupe coopérant d'organismes $AR\tilde{A}$. Ce protocole originalement conçu pour des caches coopérants, s'adapte parfaitement à une coopération asynchrone entre sites Web. Le nouveau protocole tire profit de l'aspect paresseux et non cohérent de l'échange d'informations entre les organismes pour proposer une architecture épidémique d'échange de messages.

Le protocole de communication de $AR\tilde{A}$ repose sur la communication point à point. L'échange des messages entre les membres d'un groupe coopérant ne sont pas sujets à des règles d'ordre, contrairement au protocole Relais. Les messages peuvent arriver dans des ordres différentes à chaque membre du groupe. Les incohérences possibles sont gérées par la gestion de dates (voir la section 3.7.3) et par les messages *ping* de validation des miroirs.

Messages du protocole

Le protocole est formé par un ensemble de messages contenant les informations nécessaires à la découverte des miroirs, leur disparition et leur mise à jour. L'échange d'information peut se passer de façon passif (par coopération) ou active (à la demande).

- **Échange passif**

  Nous entendons par échange passif celui où nous ne cherchons pas à être informé. Dans ce cas, l'information est acquise si un des organismes avec qui nous coopérons l'envoie de sa propre initiative. L'échange passif inclut aussi l'épidémie déclenchée par l'arrivée passive d'un message.

- **Échange actif**

  L'échange actif exige une démarche en quête de l'information. C'est à la demande de l'administrateur système qu'un message (*`LOOKUP`*) est envoyé aux autres membres du groupe pour forcer l'échange d'information sur un miroir. L'échange actif peut aussi avoir lieu lorsque l'administrateur ajoute une entrée dans la BDM et déclenche ainsi la procédure de partage de l'information avec les autres participants du groupe.

Les messages du protocole offrent une interface minimale pour informer $AR\tilde{A}$ de l'ajout d'un miroir (ou source), de son retrait ainsi que de la mise à jour d'un miroir. L'interface permet de demander aux organismes coopérants s'ils connaissent un miroir en particulier ou des miroirs d'une certaine source. Enfin, il est aussi possible de faire une requête sur l'existence des miroirs par rapport à une URL donnée.

Pour assurer une cohérence incrémentale de données stockées dans la BDM, tous les messages sont estampillés par des dates. La gestion de dates (voir la section 3.7.3) est faite par des comparaisons entre les dates connues de la cellule $AR\tilde{A}$ et celles transportées par les messages du groupe.

Formats des messages

Nous présentons ci-dessous le format des différents messages :

ADD_MIRROR
int *addMirror*(MirrorID mid)
DEL_MIRROR
int *delMirror*(MirrorID mid)
DEL_SOURCE
int *delSource*(String sup)
UPDATE_MIRROR_VERSION_BY_PREFIX
int *updateMirrorVersion*(String urlp, Date d)
UPDATE_MIRROR_VERSION_BY_MIRROR
int *updateMirrorVersion*(MirrorID mid, Date d)
FIND_MIRROR_BY_PREFIX
MirrorID *findMirrorByPrefix*(String urlp)
FIND_SOURCE
MirrorIDList *findSource*(String urlp)
LOOKUP
MirrorIDList *lookup*(String urlp)

ADD_MIRROR : int *addMirror*(MirrorID mid)

Le message *ADD_ MIRROR* notifie à tous les organismes, qui coopérent avec l'organisme local, la découverte d'un nouveau miroir 'mid'. Lorsqu'un *serveur d'organisme* reçoit le message *ADD_ MIRROR* il insère le miroir dans sa propre BDM. Si le miroir en question est déjà présent dans la BDM, le serveur vérifie la date de sa dernière mise à jour et la modifie si celle-ci est plus récente ; sinon il maintient la date. Si la date a été modifiée, le *serveur* ARÃ de l'organisme récepteur diffuse ce message aux groupes avec qui il coopère.

DEL_MIRROR : int *delMirror*(MirrorID mid)

Pour informer du retrait d'un miroir, le *serveur* ARÃ d'un organisme envoie le message *DEL_ MIRROR* à toutes les cellules avec qui il coopère. L'identificateur du miroir à supprimer est passé en paramètre. L'organisme qui reçoit le message vérifie si ce miroir fait partie de sa BDM. Si oui, il compare la date de la dernière mise à jour avec la date de la BDM, si celle-ci est plus récente que celle de la BDM le miroir est effacé ; sinon le miroir est gardé. Ensuite, si le miroir a été effacé, le *serveur* ARÃ de l'organisme récepteur diffuse ce message aux groupes avec qui il coopère.

DEL_SOURCE : int *delSource*(String urlp)

Le protocole permet le retrait de tout un ensemble des miroirs correspondant à la même source (serveur d'origine) par le message *DEL_ SOURCE*. Pour cela, le préfixe de la sous-arborescence de la source 'urlp' (ou le préfixe d'un de ses miroirs) est envoyé par l'organisme propriétaire de la source à tous les organismes avec qui il coopère. Les organismes à leur tour effaceront cette entrée de la BDM après avoir comparé la date de la disparition de la source avec la date de la création de la source dans la BDM locale. Si la date de la disparition de la source est plus récente que la date de la création de la source dans la BDM locale, l'entrée de la source est retirée de la BDM. La comparaison de la date est nécessaire dans le cas où le message a déjà été reçu par épidémie. Dans ce cas, il est possible que l'organisme propriétaire de la source ait déjà recrée un autre serveur source — cela explique une date locale plus récente. Ensuite, si l'entrée de la source dans la BDM a été éliminée, le *serveur* ARÃ de l'organisme récepteur diffuse ce message aux groupes avec qui il coopère.

UPDATE_MIRROR_VERSION_BY_MIRROR : int *updateMirrorVersion*(MirrorID mid, Date d)

Lors d'une mise à jour d'un miroir, *UPDATE_MIRROR_VERSION_BY_MIRROR* transporte l'identification du miroir (mid) et sa version (d) à tous les organismes avec qui une coopération a été établie. Les organismes à leur tour mettent à jour la date de la dernière mise à jour si celle-ci est plus récente que celle connue de leur BDM. Ensuite, si la date a été mise à jour, le message est propagé.

UPDATE_MIRROR_VERSION_BY_PREFIX : int *updateMirrorVersion*(String urlp, Date d)

Lors d'une mise à jour d'un miroir, *UPDATE_MIRROR_VERSION_BY_PREFIX* transporte le préfixe d'un url (source ou miroir) et sa version (d) à tous les organismes avec qui une coopération a été établie. Les organismes à leur tour mettent à jour la date de la dernière mise à jour si celle-ci est plus récente que celle connue de sa BDM. Ensuite, si la date a été mise à jour, le message est diffusé par le *serveur ARÃ* aux groupes avec qui il coopère.

LOOKUP : MirrorIDList *lookup*(String urlp)

Le protocole permet à un organisme de demander aux organismes coopérants s'ils ont dans leur BDM une entrée correspondante à l'URL d'un document passé à l'intérieur du message *LOOKUP*. La réponse à cette requête sera l'envoi de tout le contenu de l'entrée de la BDM : une liste des miroirs correspondant à tous les miroirs connus de cet organisme.

4.5.2 Gestion de Groupe

La gestion de groupe permet aux organismes d'entrer ou de sortir d'un groupe et assure l'échange des connaissances sur les membres d'un groupe au sein de celui-ci.

Coopérer avec un autre organisme

La coopération entre organismes ne peut avoir lieu que si les organismes se connaissent entre eux. Dans les groupes ouverts (voir la section 3.3.2), chaque organisme doit connaître au moins un membre du groupe avec qui il maintient une relation de coopération. Dans les groupes fermés (voir la section 3.3.2), tous les organismes se connaissent entre eux. C'est dans la base de données de participants (BDP) que sont maintenues les données relatives aux membres d'un groupe.

La base de données de participants (voir les section 4.3.2 et 4.7.2) est mise à jour régulièrement. Une gestion de groupe est donc nécessaire et l'objet responsable de cette gestion fournit une interface simple qui permet aux organismes d'adhérer à un groupe, quitter un groupe, etc. *ARÃ* contient un gestionnaire de groupe qui fournit les services présentés par la suite permettant le dynamisme et la flexibilité dans la coopération inter-organismes.

Abonnement

Tout organisme voulant adhérer à un groupe, doit connaître un de ses participants. Un message SUBSCRIBE est alors envoyé à cet organisme. L'organisme contacté donnera ou non l'autorisation pour que l'abonnement aboutisse. Dans ce cas, les informations à propos du nouveau participant sont diffusées parmi les membres du groupe. Ensuite, une procédure de conciliation avec le nouveau membre est déclenchée dans le but d'échanger les informations sur les miroirs connus de chacun.

- **Critères d'autorisation**

  L'organisme ayant reçu une demande d'abonnement est responsable pour le contrôle de l'entrée du nouveau membre dans le groupe. L'autorisation de l'entrée dans le groupe est donnée selon certains critères de sécurité et de taux de participation.
  Dans un *groupe ouvert* (voir la section 3.3.2), tout organisme peut faire une demande d'abonnement. Pour cela il faut juste connaître le port de demande de connexion d'un de ses participants. Les participants qui ne contribuent pas activement dans les échanges, peuvent être retirés temporairement. Cependant, chaque groupe décide de sa propre politique d'adhésion.
  Si le groupe est fermé (voir la section 3.3.2), une liste des participants est connu de tous les membres du groupe et pour rentrer dans ces groupes il faut contacter les administrateurs systèmes de chaque organisme du groupe et obtenir leurs autorisations (ou contacter l'organisme responsable de la gestion du groupe). Dans ce cas, chaque groupe choisit ses participants selon leurs propres critères.

- **Procédure de conciliation**

  La procédure de conciliation a pour but de faire connaître aux nouveaux membres la localisation des miroirs connus par le groupe. Toute la BDM, de l'organisme contacté pour l'abonnement, est alors envoyée au nouveau membre qui à son tour fera le tri dans sa propre BDM avant d'envoyer à chaque membre du groupe les informations sur les miroirs connus uniquement par lui. L'envoi de ces messages se fait par *piggybacking* ou par groupe dans un moment oisif de $AR\tilde{A}$.
  La conciliation se passe entre le nouveau membre et l'organisme contacté pour l'abonnement. Cela est possible parce que dans $AR\tilde{A}$ les BDMs d'un même groupe ne nécessitent pas d'une cohérence entre elles. Toutefois, les BDPs doivent avoir une cohérence forte dans les *groupes fermés*. Nous appelons *cohérence occasionnelle* la cohérence existante entre les BDMs d'un même groupe.

- **Cohérence occasionnelle**

  Le concept de *cohérence occasionnelle* se base sur les échanges de messages de façon paresseuse : envoi des messages par *piggybacking*, *batcher* les messages et les envoyer lorsque le serveur n'est pas chargé et par une communication asynchrone et point à point. Cette technique de coopération permet aux membres d'un groupe d'avoir accès aux mêmes *meta-données* sans pour autant garantir que tous les membres aient des BDMs identiques. Occasionnellement, les BDMs de chaque membre d'un même groupe sont identiques. Dans le cas où les BDMs ne sont pas identiques, chaque cellule $AR\tilde{A}$ peut toujours fournir le service de localisation sans que la vue de ses utilisateurs soit incohérente. La *cohérence occasionnelle* permet aux membres d'un groupe de s'enrichir avec les *meta-données* de tous les membres du groupe afin de gagner en performance sans pour autant payer le coût d'un protocole de cohérence.

Désabonnement

La procédure de désabonnement à l'intérieur des *groupes fermés* consiste à envoyer le message UNSUBSCRIBE à tous les membres du groupe. Cette procédure s'assure qu'aucun message SUBSCRIBE et UNSUBSCRIBE est en transit. L'aspect non paresseux de cette procédure vise à éviter des messages inutiles envers cet *organisme* et de optimiser la vérification des messages (SUBSCRIBE et UNSUBSCRIBE) en transit. Ensuite chaque organisme supprime le participant en question de leur base de données de participants (BDP).

Dans les *groupes ouverts* la même procédure est pratiquée mais le message UNSUBSCRIBE est envoyé aux membres connus de l'*organisme* que désabonne et non pas à tous les membres du groupe.

Coopération multiple

Un organisme peut coopérer avec plusieurs groupes. Pour cela, dans sa BDP il a une entrée pour chaque groupe auquel il appartient. La coopération entre plusieurs groupes a l'avantage de passer à l'échelle. L'idée principale étant d'avoir un protocole de coopération entre groupes d'organismes qui permet l'échange épidémique de méta-données. Ces données de localisation des miroirs sont propagées au fur et à mesure que l'épidémie croise les frontières des groupes.

La coopération multiple est possible grâce à la classe `AraGroupSet` qui connaît tous les groupes avec qui la cellule coopère. Le serveur $AR\tilde{A}$ fait appel aux méthodes de gestion de la BDM qui à travers l'`AraGroupSet` assurent la diffusion de messages en les envoyant aux serveurs $AR\tilde{A}$ de chaque *organisme*.

La coopération entre plusieurs groupes exige un système de filtrage d'information (voir la section 3.4.3) pour éviter l'inondation des messages par cycles ou par le biais des groupes qui ne s'intéressent pas à l'information. Lorsqu'un serveur reçoit un message il vérifie le *degré de contamination* (voir la section 3.4.3) de ce message et décide de la prendre ou non. En plus, la sécurité est assurée via des messages *ping* de vérification de l'existence des miroirs dans les localisations fournis dans le méta-données de la coopération.

4.6 Localisation des miroirs

$AR\tilde{A}$ localise les miroirs d'une façon transparente, grâce à un proxy qui récupère les requêtes venant du navigateur Web, tout en assurant une cohérence incrémentale avec les *sessions de cohérence*. $AR\tilde{A}$ permet l'accès au miroir le plus approprié à travers les critères de choix des miroirs et offre aussi de meilleures performances par l'interaction avec le cache.

4.6.1 Accès transparent

La localisation transparente des miroirs est un service qui permet à l'utilisateur d'accéder à une copie de la donnée sans connaître sa localisation. Dans la figure 4.1 on peut comprendre comment la localisation transparente des miroirs fonctionne :

- L'utilisateur fait une requête normale via un proxy $AR\tilde{A}$;
- Le proxy interroge le service de localisation en faisant une requête au serveur local de $AR\tilde{A}$ ($AR\tilde{A}server$) ;

- Le service de localisation cherche dans la BDM l'existence d'un miroir et retourne le résultat au proxy ;
- Le *$AR\tilde{A}$proxy* redirige vers la localisation retournée.

Le coût de la redirection de la requête dans $AR\tilde{A}$ est local. *$AR\tilde{A}$proxy* se communique avec *$AR\tilde{A}$server* à travers l'ouverture d'une connexion pour chaque navigateur Web. À chaque requête, *$AR\tilde{A}$proxy* envoie un message à *$AR\tilde{A}$server* avec une demande de recherche de localisation d'un serveur Web pour un URL donné. Lorsque *$AR\tilde{A}$proxy* connaît l'adresse du serveur le plus approprié qui contient l'URL demandé par l'utilisateur, il donne suite à la requête dans le Web.

Le *$AR\tilde{A}$proxy* est responsable aussi de l'interaction avec le cache. Le cache est le premier à être interrogé par le *$AR\tilde{A}$proxy*. Si le document ne se trouve pas dans le cache, *$AR\tilde{A}$proxy* interroge alors le serveur $AR\tilde{A}$. Celui-ci répond avec une liste des miroirs ou avec un NULL. Si la réponse contient une liste des miroirs, *$AR\tilde{A}$proxy* reinterroge le cache sur les URLs contenus dans la liste. Enfin, si aucune de ces recherches aboutie, *$AR\tilde{A}$proxy* va chercher le document chez le serveur source.

Nommage

La rédirection de la requête est possible grâce au système de nommage utilisé par $AR\tilde{A}$ et sa technique de recherche.

Le nommage de réplicats est résolu à travers une table de correspondance entre le nom de la source et les noms des miroirs de cette source. La table de correspondance est matérialisé avec la BDM qui contient les données suivantes :

1. Prefixe de l'URL de la source
2. Liste de préfixes des l'URLs des miroirs de la source

La recherche dans la BDM est capable de retrouver le préfixe le plus long (voir la section 4.7.1). Cela permet de retrouver tous les miroirs qui peuvent satisfaire la requête de l'utilisateur. La redirection dans $AR\tilde{A}$ profite donc d'une plus grande disponibilité grâce à cette technique de recherche qui prend en compte les sous-miroirs d'un serveur source.

4.6.2 Choix du miroir

Plusieurs études ont été menées [MDZ98, BAZ$^+$97, CC96b, PMM93] dans le sens d'optimiser l'accès aux données. Nous avons choisi nos critères basés sur ces recherches.

Temps d'accès

$AR\tilde{A}$ calcule le temps d'accès à un miroir par l'envoi des messages *Ping* (ces messages servent aussi à valider l'existence d'un miroir). $AR\tilde{A}$ envoie des messages *ping* périodiquement à des horaires diverses et à différents jours de la semaine. La BDM du système $AR\tilde{A}$ garde une valeur représentatif des mesures faites avec les messages *ping*. Cette valeur est le résultat d'une moyenne pondérée, calculée de manière à tenir compte des dernières mesures et de l'historique des mesures.

Le calcul du temps d'accès prend aussi en considération le nombre des routeurs entre l'organisme $AR\tilde{A}$ et le miroir en question [Tac97]. En utilisant la commande `traceroute`, nous utilisons le nombre des routeurs pour calculer l'ordre croissant de temps d'accès vers les serveurs. Les informations issues des routeurs et des *ping* sont combinées pour établir l'ordre des serveurs par temps d'accès.

Version

La BDM du système $AR\tilde{A}$ connaît pour chaque miroir la date de sa dernière mise à jour (version). Cette donnée est échangée entre les organismes coopérants. Quand un organisme prend connaissance d'une version plus récente d'un miroir, il fait connaître à ses partenaires (les membres des groupes avec qui il coopère) l'existence de cette nouvelle version en leur transmettant la localisation du miroir correspondant.

Cette connaissance des versions sur les miroirs peut être utilisée pour une éventuelle mise à jour des miroirs. Ces miroirs se mettraient à jour non pas en cherchant les données à la source mais en les cherchant sur un autre miroir moins coûteux en latence et/ou bande passante et limitant ainsi la charge du serveur source.

Charge du serveur

Dans la version actuelle, $AR\tilde{A}$ n'utilise pas la charge du miroir comme critère de choix. Dans des travaux futurs nous envisageons de prendre en compte la charge en étudiant le nombre d'utilisateurs et de requêtes sur un miroir.

4.6.3 Sessions de cohérence

$AR\tilde{A}$ assure une vue avec une cohérence incrémentale par l'utilisation d'une *session de cohérence*. La réalisation d'une *session de cohérence* dans $AR\tilde{A}$ utilise une technique qui tient compte de la version de chaque serveur (miroir) par rapport à la vue de l'utilisateur.

Pour signaler le serveur de $AR\tilde{A}$ l'existence ou non d'une *session de cohérence*, l'utilisateur choisit explicitement d'ouvrir une session. Pour contrôler les sessions, $AR\tilde{A}$ crée une *variable session* pour chaque navigateur Web connecté au proxy $AR\tilde{A}$. Ces *variable session* peuvent prendre la valeur TRUE ou FALSE selon que l'utilisateur désire la garantie du *non retour sur le passé* ou non. Lors de l'établissement de la connexion, la *variable de session* est affectée avec la valeur FALSE.

Ouverture et fermeture d'une session

L'ouverture d'une session se fait en signalant le navigateur Web avant la première requête à travers une *variable de session* mise à TRUE. Cette variable peut être levé (mise à TRUE) pendant toute la durée de la connexion.

La fermeture d'une *session de cohérence* se fait en signalant le navigateur Web du fin de la session à travers l'affectation de la *variable de session* à la valeur FALSE. La session peut être fermée à tout moment pendant une connexion.

Vecteur de versions

$AR\tilde{A}$ utilise des *vecteurs de versions* [JPR+83] pour garantir une cohérence incrémentale. Les *vecteurs de versions* ne gardent pas la date du document mais celle du serveur. La date d'un serveur couvre celles des documents qu'il contient. Pour pouvoir assurer le *non retour sur le passé*, le serveur de $AR\tilde{A}$ a besoin de connaître la version de tous les serveurs pouvant satisfaire la requête de l'utilisateur. Le serveur de $AR\tilde{A}$ a aussi besoin de connaître la version du serveur actuel qui répond à la requête. $AR\tilde{A}$ a un vecteur courant par session. $AR\tilde{A}$ maintient pour chaque session de l'utilisateur un *vecteur de versions* pour garder les versions de tous les miroirs d'un même serveur, ainsi que la version du serveur en cours d'accès.

Le *vecteur de versions* contient dans le premier champ la date de la version du serveur en cours d'accès (*date référence*). Les autres champs correspondent aux dates des versions de tous les serveurs qui répliquent le même objet et connu de la BDM. Le format d'un *vecteur de versions* est le suivant :

Vecteur[dvsCA, dvsR1, ..., dvsRn]

- **dvsCA** : **d**ate de la **v**ersion du **s**erveur en **C**ours d'**A**ccès
- **dvsRi** : **d**ate de la **v**ersion du **s**erveur **R**épliqué **i**

où **n** est le nombre des serveurs (miroirs) qui répliquent le même objet que le serveur en cours d'accès et **dvsCA** est la *date référence* (**dRef**) pour la cohérence incrémentale : **dRef = dvsCA**

Conditions pour garantir la *cohérence incrémentale*

La cohérence incrémentale est assurée si la date de la version du serveur i à accéder est supérieure ou égal à la date de la version du serveur en cours d'accès :

dvsCA $\leq$ dvsRi

Au moment de changer le serveur d'accès, $AR\tilde{A}$ vérifie cette condition. Au cas où cette condition n'est pas satisfaite, $AR\tilde{A}$ choisit d'accéder au serveur source (qui contient toujours la version la plus récente).

Dans le cas où plusieurs serveurs satisfont la condition de cohérence, $AR\tilde{A}$ tient compte du temps d'accès pour choisir le serveur.

Granularité

Le système $AR\tilde{A}$ travaille avec de gros grain (le miroir) pour assurer une cohérence incrémentale à la demande (*session de cohérence*). Le choix de travailler avec de gros grain repose sur la simplicité de gérer (stocker et choisir) les versions et sur le peu de messages échangés pour les connaître.

Le grain étant le serveur lui même, le nombre de versions à connaître est égal au nombre des serveurs connus de chaque cellule $AR\tilde{A}$. Chaque *vecteur de versions* a donc **n+1** champs ; où **n** est le nombre des serveurs non accédés.

Cette technique permet de choisir rapidement lors de l'ouverture d'une session, un serveur incrémentalement à jour.

Fausse incohérence

Le fait de travailler avec un gros grain, peut faire gagner en performance mais peut diminuer la disponibilité. Les serveurs qui remplissent la condition de cohérence sont un sous-ensemble du nombre total des serveurs connus par $AR\tilde{A}$.

Quand un serveur devient indisponible parce qu'il ne satisfait pas la condition de cohérence imposée ; il est possible que certains documents dans ce serveur remplissent la condition de cohérence isolément. En fait, les documents en lecture par l'utilisateur sont un sous-ensemble des documents disponibles dans ce serveur et peuvent tout à fait être à jour par rapport aux autres copies. Nous appelons *fausse incohérence* le fait d'exclure un serveur de l'ensemble de serveurs disponibles et cohérents alors que des documents de ce serveur présentent une version cohérente.

4.6.4 L'interaction avec le cache

L'interaction avec le cache peut causer de problèmes d'incohérence. Pour faire face à cette situation, $AR\tilde{A}$ tient compte de la collaboration avec le cache dans la gestion de *sessions de cohérence.*

Dans les caches le grain de gestion de versions est le document, contrairement à la gestion proposée par $AR\tilde{A}$ qui est basée sur les serveurs. C'est à ce niveau que les conditions de cohérence incrémentale seront posées.

Conditions pour garantir la cohérence incrémentale

Dans le cas d'un accès à un document dans le cache, la date du document dans le cache doit être égale ou plus récente que la *date référence* en cours pour que la cohérence incrémentale soit assurée :

$$\mathbf{dRef} \leq \mathbf{dvDC}$$

où **dRef** est la *date référence* et **dvDC** est la date de la version du document dans le cache.

Lorsqu'un document est inséré dans le cache, la *date référence* bascule de la date de la version d'un serveur vers la date de la version d'un document :

$$\mathbf{dRef} = \mathbf{dvDC}$$

L'accès aux documents dans le cache peut diminuer considérablement le temps de réponse à l'utilisateur. Cependant, le changement de grain dans le contrôle de la version peut diminuer la disponibilité des documents cachés. Si pour un document la version gardée est celle de la dernière modification du document, il est probable que la *date référence* soit plus récente et que le document ne remplisse donc pas la condition de cohérence. En effet, la *date référence* d'un serveur tient compte de la mise à jour de tous les documents du serveur. Par contre, si pour chaque document trouvé dans le cache, la date de version retenue est la date du serveur de ce document, l'effet « changement de grain » disparaît.

4.7 Structures de données

Le système $AR\tilde{A}$ utilise deux base de données, la première pour assurer le stockage des données sur la localisation des miroirs et la deuxième pour garder l'identification des *organismes* membres de tous les groupes avec lesquels la *cellule* coopère.

4.7.1 Base de Données de Miroirs

L'accès transparent aux miroirs passe par la création d'une base de données des divers miroirs liés à la même source. Le préfixe de la sous-arborescence de l'URL source est l'identificateur unique de tous les miroirs répliqués. Pour chaque serveur 'miroré' nous établissons une liste de préférence des miroirs ainsi que la date de sa dernière mise à jour (les champs *best* et *version* dans la figure 4.5).

C'est cette base de données qui permet la recherche des miroirs par rapport à une URL donnée. Il y a une base de données de miroirs (BDM) dans chaque cellule (organisme).

La BDM peut être accédée de trois façons différentes : par l'administrateur système, par un autre organisme coopérant et par le proxy quand le navigateur Web émet une requête. L'administrateur système peut invoquer toutes les méthodes de l'interface. Les autres organismes coopérants accèdent indirectement à l'interface par l'intermédiaire d'un serveur et $AR\tilde{A}$ local. Ces dernières, ne peuvent pas faire de mise à jour sur la métrique d'un miroir (liste de préférences des miroirs) ; ce concept étant propre à chaque cellule en particulier.

Le navigateur Web accède à l'interface BDM à travers d'un HttpProxy et du serveur $AR\tilde{A}$ local. Ces objets assurent à l'utilisateur un accès transparent aux miroirs répertoriés dans la BDM local ; les méthodes utilisées étant celles qui consultent le contenu de la BDM uniquement.

identificateur unique	identificateur miroir	*best*	version
prefixe de l'URL source de la sous-arborescence	prefixe de l'URL miroir de la sous-arborescence	entier	date de la derniere mise-a-jour
source 1	miroir 1	n1	date 1
	miroir 2	n2	date 2
	miroir 3	n3	date 3
	⋮	⋮	⋮
source 2	miroir 1	n1	date 1
	miroir 2	n2	date 2
	miroir 3	n3	date 3
	⋮	⋮	⋮
⋮	⋮	⋮	⋮
source n	miroir 1	n1	date 1
	miroir 2	n2	date 2
	miroir 3	n3	date 3
	⋮	⋮	⋮

FIG. 4.5 – *Base de données de miroirs*

Correspondance par préfixe

La correspondance par préfixe est assurée par une table de hachage où l'entrée contient l'URL de la source (voir la section 3.5.1) lié à une liste des URLs des miroirs (voir figure 4.5).

La recherche du préfixe. La recherche par préfixe se fait à travers d'une méthode de comparaison incrémentale sur la chaîne de caractères de l'URL. La recherche se fait dans les deux sens. D'abord $AR\tilde{A}$ cherche l'URL dans le champ *identificateur unique*. Si $AR\tilde{A}$ trouve l'URL correspondante, $AR\tilde{A}$ retourne le miroir qui a le plus petit entier dans le champ *best* ou la *version* la plus récente. Si $AR\tilde{A}$ ne trouve pas l'URL correspondante, $AR\tilde{A}$ cherche cet URL parmi la liste des miroirs. Si la recherche est positif, $AR\tilde{A}$ utilise les mêmes critères expliqués ci-dessus pour choisir les miroirs à retourner.

Les sous-miroirs. La version actuelle de $AR\tilde{A}$ peut trouver des sous-miroirs mais n'a pas été optimisée pour faire la recherche en tenant toujours compte de sous-miroirs en utilisant un arbre binaire léxicographique (voir la section 3.5.1). La recherche des sous-miroirs est importante pour profiter de tous les réplicats pouvant satisfaire une requête. Le sous-miroirs sont localisés par une recherche de plus long préfixe dans la table de hachage.

Si nous avons un URL

http ://www.mit.edu/systems/publi/index.html

des comparaisons successives sont effectués. D'abord avec le chemin complet de l'URL :

http ://www.mit.edu/systems/publi

ensuite

http ://www.mit.edu/systems

et enfin

http ://www.mit.edu

Dans le cas où la BDM contient deux entrées : une entrée pour les miroirs de tout MIT et autre entrée pour les miroirs de la partie système du MIT, notre recherche trouve d'abord le sous-miroir de la partie système.

Critères de choix du miroir

Dans la BDM nous avons deux champs qui permettent de choisir un miroir parmi d'autres. Le premier garde le temps d'accès à un site : c'est le champ *best*. Le deuxième garde la date de mise à jour du miroir. Dans le champ *best* on garde une estimation du temps d'accès pour chaque miroir (ce champ peut aussi être utilisé comme un classement des miroirs par un autre critère). Le champ *version* garde la date de la dernière mise à jour de chaque miroir. Actuellement, ce champ garde la date à laquelle le miroir a été catalogué.

Interface d'accès

L'interface d'accès de la base de données de miroirs (BDM) permet la consultation de cette base et sa modification. Ainsi, nous offrons des méthodes (voir l'annexe A) pour consulter la base afin d'identifier l'existence d'un miroir, ajouter un miroir afin d'alimenter la base de données, modifier le contenu d'un miroir pour maintenir sa date de dernière mise à jour, et enfin effacer un miroir à la suite de sa disparition.

4.7.2 Base de Données de Participants

Chaque organisme souhaitant coopérer avec d'autres organismes, doit connaître les participants de son groupe de coopération. Pour cela, une base de données contenant tous les organismes avec lesquels la coopération a lieu est maintenue. Cette base de données est particulière à chaque organisme ce qui permet à chaque organisme d'avoir son propre groupe de coopération différent des autres.

Cette base de données de participants (BDP) permet au protocole de coopération de savoir l'adresse IP de chaque organisme (serveur) avec qui une collaboration a été établie, le nom de

identificateur du groupe (entier)	adresse IP de l'organisme	nom de l'organisme	port IP de l'organisme
groupID 1	addrIP 11	name 11	portIP 11
	addrIP 12	name 12	portIP 12
	addrIP 13	name 13	portIP 13
	⋮	⋮	⋮
groupID 2	addrIP 21	name 21	portIP 21
	addrIP 22	name 22	portIP 22
	addrIP 23	name 23	portIP 23
	⋮	⋮	⋮
⋮	⋮	⋮	⋮
groupID n	addrIP n1	name n1	portIP n1
	addrIP n2	name n2	portIP n2
	addrIP n3	name n3	portIP n3
	⋮	⋮	⋮

FIG. 4.6 – *Base de données de participants*

l'organisme et le port IP (voir figure 4.6). Les messages sont adressés à tout le groupe selon l'algorithme de filtrage choisi par chacun des participants.

Le service de nommage est spécialisé pour le support de la communication de groupe. La BDP maintient des liste d'associations (<identificateur du groupe>[<adresse IP><URL><port>]+).

L'identificateur du groupe est un entier identifiant de façon unique le groupe. Chaque tuple [<adresse IP><URL><port>] représente les coordonnées d'un membre de ce groupe.

Interface d'accès

L'interface d'accès de la base de données de participants (voir l'annexe B) permet la consultation de cette base et sa modification. Nous offrons des services pour consulter la base afin d'identifier l'existence d'un organisme ou un groupe d'organismes, ajouter un organisme ou un groupe, et effacer un organisme ou un groupe.

4.8 Conclusion

Nous avons présenté dans ce chapitre la réalisation du prototype $AR\tilde{A}$. Le système $AR\tilde{A}$ a été implémenté en Java et est donc entièrement orienté objet. $AR\tilde{A}$ a une modélisation permettant des choix flexibles et dynamiques à niveau de la gestion de groupes et du choix du miroir.

Notre solution repose sur les éléments suivants : (i) la coopération entre cellules ; (ii) la communication paresseuse et épidémique ; (iii) la table de corrélation entre URLs ; (iv) l'interaction avec le cache ; (v) des sessions pour garantir une cohérence incrémentale [TDP+94].

L'architecture proposée repose sur la coopération entre cellules. La coopération entre les organismes est basée sur des liaisons point-à-point et repose sur l'échange épidémique des messages d'une façon paresseuse.

La solution proposée permet de prendre en compte plusieurs critères pour choisir le miroir le plus approprié à accéder. Nous avons souligné l'aspect flexible dans la réalisation qui permet à chaque cellule $AR\tilde{A}$ de calculer et stocker le(s) critère(s) désirés. En outre, l'utilisateur lui-même pourra indiquer dans sa requête le critère le plus important pour lui.

La localisation est réalisé de manière transparente grâce à un proxy qui rédirige la requête http vers $AR\tilde{A}$. Le système $AR\tilde{A}$ assure alors la rédirection vers le miroir le plus approprié. Le coût de cette rédirection est local à l'*organisme* grâce à la BDM (Base de Données de Miroirs) qui stocke au fur et à mesure la localisation des miroirs.

Le prototype $AR\tilde{A}$ offre toutes les propriétés de sa spécification. Cependant, certaines stratégies de choix du miroir n'ont pas été implémentées par manque de temps. En outre, nous avons traité le problème de la cohérence par le concept des sessions. L'ouverture d'une session garantit le « non retour sur le passé » même avec l'interaction avec le cache.

Troisième partie

Éléments de validation

Chapitre 5

Méthodologie d'évaluation

Sommaire

5.1 Introduction

Nos mesures visent à comprendre le comportement des requêtes Web afin de vérifier si la création des miroirs peut améliorer le temps de réponse ainsi que diminuer le trafic réseau. En outre, ces mesures permettent de vérifier l'intérêt de l'échange d'informations sur la localisation des miroirs entre organismes.

La méthodologie adoptée est composée de trois étapes comme on peut le voir dans la figure 5.1 . La première étape consiste à collecter des traces sur le trafic réseau dans l'organisme. Ensuite ces traces sont analysées de façon interne et externe (par rapport à d'autres organismes). L'analyse interne étudie le trafic d'un organisme par rapport au nombre de requêtes par serveur et le nombre d'octets par requêtes et par serveur. Cette étude donne comme résultat une liste des candidats serveurs à être *mirorés*. L'analyse externe permet d'évaluer l'intérêt de la coopération pour la localisation de ces serveurs candidats.

Ensuite, le temps d'accès à ces serveurs est mesuré et une configuration de placements de miroirs (local, métropolitain, national, continental) est proposée. Les traces sont alors réanalysées en fonction de la configuration. Les deux étapes *analyse* et *configuration* se succèdent pour raffiner et valider la solution proposée. On étudie alors la diminution du trafic réseau ainsi que du temps de réponse en fonction du placement des miroirs.

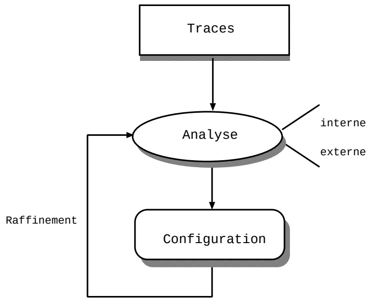

FIG. 5.1 – *Méthodologie pour l'étude des traces*

5.2 Collecte de traces

Les traces utilisées pour cette étude ont été collectées au niveau de trois organismes : l'INRIA, l'INT et l'ENST (voir fig. 5.2.1). Cette collecte a été réalisée par Guillaume Pierre [Pie99] et, de ce fait, elle date à peu près de deux ans. Le fait que les traces soient relativement anciennes ne change en rien le fond de notre étude ni ces résultats puisqu'elle est basée sur des scénarios fictifs qui miment ce qui se passe sur le Web au niveau du miroring.

5.2.1 Les informations collectées

La collecte des traces a été réalisée à l'aide de proxy caches placés dans les trois organismes. Les données collectées comportent l'ensemble des requêtes des utilisateurs qui sont passées par ces caches. Ceci exclut les requêtes filtrées par les caches personnels.

Les traces contiennent les informations suivantes : l'adresse du client, l'URL du document demandé, la date de la requête, la taille du document et la version du document délivré (champ HTTP `last modified`). Ces informations sont suffisantes pour l'étude des caractéristiques internes et externes du trafic. Cependant, elles sont insuffisantes pour l'étude de la cohérence incrémentale garantie par $AR\tilde{A}$ et son impact sur la localisation.

Afin d'évaluer la performance d'un système de localisation de miroirs en terme de cohérence des sites choisis par rapport aux serveurs d'origine, un simulateur doit comparer, pour chaque requête envoyée, la version délivrée au client et celle qui aurait été délivrée par le serveur d'origine. Or la version du serveur d'origine n'est pas tout le temps délivrée par le proxy cache. En cas de *miss*, le cache récupère le document sur le serveur et la version obtenue est égale à la version idéale. Par contre, en cas de *hit*, le cache délivre le document qu'il contient et ainsi la version du serveur d'origine à ce moment n'est pas connue. Alors, [Pie99] a rejoué ces requêtes ultérieurement pour obtenir cette information importante.

	Nb de requêtes	Début Collecte	Fin Collecte
INRIA	3440847	12/8/1998	3/1/1999
INT	2593281	23/11/1998	15/2/1999
ENST	236436	13/9/1998	8/10/1998
Total	6270564		

TAB. 5.1 – *Informations sur les traces collectées*

5.3 L'analyse interne des traces

L'analyse interne des traces vise à comprendre les traces d'un *organisme* en particulier pour montrer l'intérêt de construire des miroirs et quels miroirs construire. Il faut donc identifier les serveurs intéressants à être mirorés, appelés *miroirs potentiels* ou « candidats ».

5.3.1 Intérêt des miroirs

Est-il intéressant de mirorer ? Pour répondre à cette question, nous nous sommes intéressés au nombre des requêtes faites par serveur dans une période donnée, ainsi qu'au nombre d'octets téléchargés par serveur. La première mesure nous informe sur le nombre de connexions [BL95] établies avec chaque serveur. Chaque requête a un coût à payer égal au coût d'une ouverture de connexion. De ce fait, plus il y a des requêtes sur un serveur, plus est coûteux l'accès vers ce serveur. La deuxième mesure montre le trafic géré par chaque serveur. Le coût du trafic va dépendre des ressources réseau utilisées pendant le transfert.

5.3.2 Choix des candidats potentiels

Pour savoir s'il est intéressant de *mirorer* un serveur, nous considérons une liste des serveurs « candidats » à être *mirorés*. Cette liste est composée des serveurs les plus accédés et qui ont le plus de trafic.

Dans un premier temps, on s'intéresse à la localité des requêtes par serveur. On comptabilise alors le nombre des requêtes qui accèdent au même serveur et on s'intéresse aux serveurs les plus accédés. Le but ici est de trouver des serveurs candidats à « être mirorés ». La création d'un miroir peut être motivée par diverses raisons. Dans notre étude, les paramètres considérés sont le trafic réseau et le temps de retrait d'un document. En principe, un serveur devrait « être miroré » si le nombre des requêtes est tel que la création d'un miroir proche peut diminuer d'une part le trafic réseau, d'autre part le temps de réponse. On se contentera pour commencer par sélectionner les serveurs le plus accédés.

Le nombre des requêtes en soi n'est pas une mesure suffisante pour la création d'un miroir. Pour que le gain en trafic réseau et en temps de réponse soit significatif après la création d'un miroir, encore faut-il que le miroir se trouve plus proche que le serveur lui même. Comprenons le terme « proche »comme correspondant à un site qui offre un meilleur temps d'accès. Dans ce sens, on doit aussi mesurer le temps d'accès vers chaque serveur. Les serveurs qui présentent le plus grand temps d'accès seront retenus comme candidats potentiels à « être mirorés ».

À partir de deux listes de candidats, l'une classée par trafic réseau, l'autre par temps d'accès, une autre liste qui tient compte de ces deux mesures à la fois peut être définie. On élimine de la première liste les serveurs qui ont un temps d'accès suffisamment modéré. De même, pour la deuxième liste, on élimine les serveurs qui ont trop peu de requêtes. On possède alors une liste des candidats à « être mirorés ».

5.3.3 Les mesures effectués

Dans un premier temps nous avons mesuré le pourcentage des requêtes en fonction du pourcentage de serveurs accédés. Comme montre la figure 5.2, la plupart des requêtes s'adressent à moins de 10% des serveurs dans les trois organismes étudiés. Cela signifie que d'une manière générale le trafic se passe dans une minorité de serveurs.

Afin de vérifier si le trafic se restreint à une minorité de serveurs, nous avons mesuré le pourcentage du volume des données par pourcentage de serveurs accédés. Encore une fois, nous

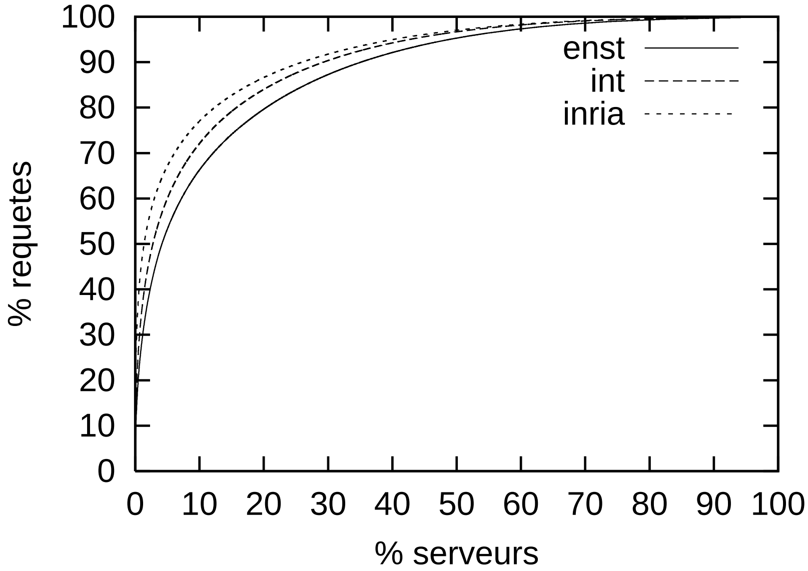

FIG. 5.2 – *Le nombre de requêtes par serveur*

observons dans la figure 5.3 que la plupart du volume des données provient de moins de 10% des serveurs dans les trois organismes étudiés. En conclusion, nous affirmons qu'environ 10% des serveurs sont responsables pour plus de 70% du trafic et des requêtes. De ce fait, si la latence vers ces serveurs diminue, nous pouvons espérer un gain important en temps de réponse et en bande passante. Le tableau 5.4 renforce l'idée que le coût à payer pour l'accès Internet se concentre sur une petite pourcentage des serveurs. Par exemple, à l'INRIA, parmi les serveurs responsables pour 70% des requêtes (6%) la moitié font aussi parti des serveurs responsables de 70% du trafic (3%). La question qui se pose est de savoir si les serveurs communs entre ces deux listes sont toujours responsables pour une partie significative du trafic et du nombre des requêtes. Le tableau 5.5 montre le pourcentage du trafic et le pourcentage de requêtes attribués à ces serveurs en commun. Nous vérifions que ces serveurs sont responsables de près de la moitié du trafic et des requêtes. Nous constatons ainsi que ces serveurs forment une liste de candidats potentiels à être *mirorés*.

5.4 Analyse externe des traces

L'analyse externe des traces étudie la localité d'accès entre les différents *organismes* par rapport aux *miroirs potentiels*. Le but de cette étude est de comprendre le réel intérêt de lc coopération dans un système de localistion des miroirs.

5.4.1 Localiser les miroirs

Afin d'accéder un miroir, il faut tout d'abord connaître son existence et sa localisation. Un système de localisation et de découverte de miroir peut avoir différents approches : (i) une approche global (une solution à la DNS : LDS [KRR99]) ; (ii) une approche régionale (une solution coopérative : *ARÃ [Dor99]*) ; (iii) un moteur de recherche [BB99], etc.

Nous voulons savoir quel système de découverte et de localisation des miroirs serait le plus intéressant compte tenu des gains que nous pouvons avoir. Est-il intéressant le coût de connaître

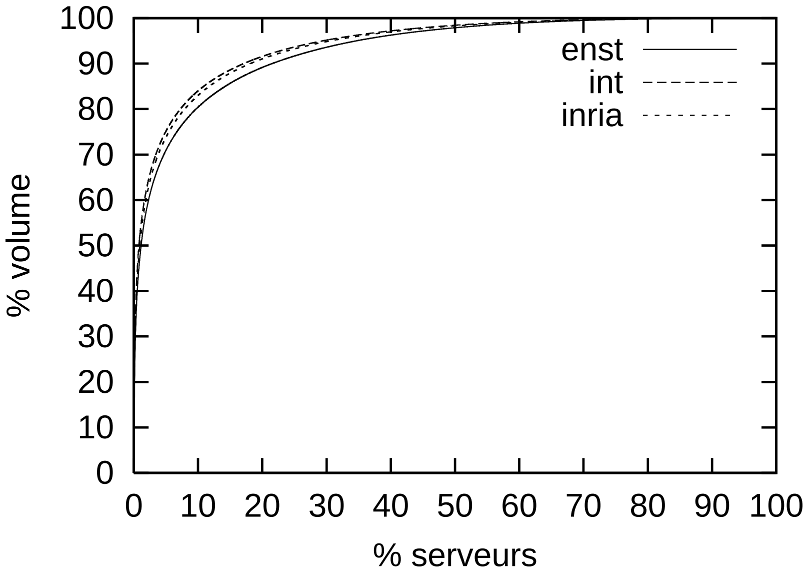

FIG. 5.3 – *Le nombre d'octets par serveur*

	serveurs pour 70% req.	serveurs pour 70% d'octets	commun
ENST	890/7342 = **12%**	343/7342 = **4%**	247/7342 = **3%**
INRIA	2926/47787 = **6%**	1815/47787 = **3%**	1214/47787 = **2%**
INT	4275/48186 = **8%**	1648/48186 = **3%**	1078/48186 = **2%**

FIG. 5.4 – *Les serveurs qui ont beaucoup des requêtes et de trafic*

tous les miroirs ? Les mesures faites montrent que la plupart du trafic concerne un nombre limité des serveurs. Cela nous fait croire que le coût d'un système de localisation global n'est pas justifiable. Vaut-il le coût d'échanger avec d'autres organismes les informations concernant la localisation des miroirs ? Pour répondre à ces questions, nous procédons à deux mesures. La première consiste à trouver la localité d'accès entre les organismes, i.e la localité entre groupes d'utilisateurs différents qui partagent les mêmes intérêts. Nous voulons savoir si le nombre des serveurs communs accédés par ces utilisateurs est significatif pour justifier une coopération entre organismes. Ensuite nous mesurons le nombre des candidats à miroir d'un organisme qui sont accédés par les autres organismes. Si le pourcentage de candidats à miroir qui profite aux autres organismes est élevé, nous pouvons supposer qu'une coopération entre organismes est une bonne solution.

	commun	% requetes	% trafic
ENST	3%	39,67%	45,83%
INRIA	2%	53,58%	50,79%
INT	2%	40,21%	42,95%

FIG. 5.5 – *Les pourcentage de trafic et des requêtes pour les serveurs communs*

	total serveurs			
ENST	7342	7342		7342
INRIA	47787		47787	47787
INT		48186	48186	48186
localite	9%	9%	36%	7%

FIG. 5.6 – *Localité entre organismes*

5.4.2 Localité d'accès entre organismes

Le tableau 5.6 montre le pourcentage des serveurs communs entre les différents organismes. Pour effectuer ces mesures, nous avons considéré tous les serveurs accédés dans chacune des traces. Nous observons que la localité entre l'INRIA et l'INT de 36% est beaucoup plus élevé que la localité trouvé entre les autres organismes. Cela est dû au fait que le nombre total des serveurs accédés par l'ENST est beaucoup plus petit que celui accédé par l'INRIA et par l'INT. La localité entre les trois organismes (ENST/INRIA/INT) est de 7%. Les chiffres sont tout de même satisfaisants, compte tenu du fait que le gros du trafic concerne moins de 10% du total des serveurs.

Pour que la localité entre les organismes soit significative, encore faut-il que les serveurs communs soient responsables de la plupart du trafic. Le tableau 5.7 affiche le pourcentage des candidats à miroir de chaque organisme qui sont aussi accédés par un autre. Ici, on croise la liste des candidats à miroir d'un organisme avec le fichier contenant tous les domaines accédés par un

localite de tous les domaines de la trace --------------- candidats a miroir		Tous les domaines de la trace		
		ENST	INRIA	INT
candidats a miroir	ENST	100%	80%	85%
	INRIA	52%	100%	81%
	INT	50%	83%	100%

FIG. 5.7 – *Localité par rapport aux candidats à miroir*

autre. Le taux de localité dans ce cas est important. Le résultat montre que si l'information sur la localisation de ces « miroirs » est échangée entre ces organismes le profit qui en est tiré doit être positif.

localite candidats a miroir		candidats a miroir		
		ENST	INRIA	INT
candidats a miroir	ENST	100%	48%	50%
	INRIA	9%	100%	34%
	INT	11%	38%	100%

FIG. 5.8 – *Localité entre les candidats à miroir*

Le tableau 5.8 confirme le taux élevé de localité entre les serveurs responsables de la plus grande partie du trafic. Ici, nous avons procédé à une comparaison entre les candidats à miroir de chaque organisme. Le calcul de pourcentage est fait par division à chaque fois par un de deux organismes en question. Nous pouvons donc comparer le profit potentiel d'un organisme par rapport à l'autre.

cette étude montre l'intérêt potentiel de la coopération. En effet, les organismes étudiés présentent un taux élevé de localité en ce qui concerne les serveurs les plus accédés et les serveurs responsables de la plus part du volume des données transférées. Cela justifie une politique d'échange d'information sur la localisation des miroirs. Cependant, il faut tenir compte du coût réseau imposé par la coopération entre les organismes. Une comparaison entre le gain obtenu avec le placement des miroirs et le coût de la coopération permettra de conclure sur les bienfaits de la coopération.

5.5 Conclusion

Dans ce chapitre, nous avons étudié les traces de trois organismes différents par rapport à deux points de vue : l'analyse interne et l'analyse externe.

Dans un premier temps, l'analyse interne nous a montré l'intérêt de construire des miroirs. Dans un deuxième temps, elle nous a permis de constater que le pourcentage des serveurs vraiment sollicités est inférieur à 10%. Ces résultats nous ont menés vers une solution non globale de localisation des miroirs sur le Web.

L'analyse externe a montré l'intérêt de la coopération entre ces organismes. Tout d'abord, nous avons vérifié que le taux de localité entre les différents organismes est significatif. Ensuite, nous avons validé ce taux de localité par rapport aux serveurs candidats à être mirorés.

Ces deux analyses nous ont guidés vers la conception et la réalisation d'un système de localisation des miroirs par coopération, ARA. Dans le chapitre suivant, nous evaluons le gain et le coût d'$AR\tilde{A}$ pour la localisation de ces miroirs fictifs.

Chapitre 6

Mesures de performance

Sommaire

6.1 Introduction

Dans ce chapitre, nous nous penchons sur l'évaluation des principaux systèmes de localisation en utilisant les traces décrites dans le chapitre 5. Nous commençons par définir ces systèmes. Puis, nous dressons une liste générale de métriques nécessaires pour la comparaison de systèmes de localisation, exprimant les gains et les coûts engendrés par un système donné. Ensuite, nous définissons des scénarios correspondants à des placements de miroirs purement *hypothétiques*, mais néanmoins plausibles. Nous rejouons les traces des différents scénarios pour les différents systèmes de localisation choisis et nous les comparons suivant les métriques définies pour en déduire l'utilité de la localisation de miroirs ainsi que le système de localisation le plus adapté.

6.2 Systèmes de localisation considérés

Ci-dessous, nous définissons les différents systèmes de localisation que nous considérons pour notre étude comparative par la suite (voir section 6.2).

- **personnel** : chaque utilisateur découvre les miroirs qui l'intéressent et ne partage cette information avec personne ;
- **local** : un groupe d'utilisateurs partage l'ensemble des miroirs découverts ;
- **coopératif** : un ensemble d'organismes partage l'ensemble des miroirs découverts mutuellement ;
- **idéal** : tous les miroirs sont connus à l'avance (aucun coût de découverte) et partagés par tout le monde

6.3 Métriques de comparaison

Dans cette section, nous présentons l'ensemble des métriques utilisées pour l'évaluation des différents systèmes de localisation introduits dans la section précédente. Cet ensemble de métriques constitue un noyau minimum des métriques à explorer pour tous les systèmes de localisation. Il devrait être complété en fonction des propriétés des protocoles utilisés (i.e, ajouter une métrique sur la cohérence au cas où l'un des protocoles en propose).

Dans ce qui suit, Un « hit » correspond à une requête qui a été indiquée par le système de localisation. Un « miss » correspond à une requête dont la localisation n'a pas pu être indiquée par le système de localisation.

6.3.1 Amélioration du taux de hit

Cette mesure exprime l'amélioration du taux de hit par rapport à une configuration sans système de localisation. En d'autres termes, elle exprime l'augmentation du taux de miss qui aurait été enregistrée en l'absence d'un système de localisation. Cette dernière correspond au pourcentage du nombre de requêtes servies par un système de localisation sur le nombre de requêtes qui auraient pu être servies par le système de localisation idéal (les requêtes mirorées).

$$\uparrow \%\ Hits = \frac{Nb\ de\ requetes\ localisees}{Nb\ de\ requetes\ mirorees} * 100$$

6.3.2 Gain global en temps

Cette mesure exprime l'amélioration en temps de retrait sur l'ensemble des requêtes mirorées par rapport à une configuration sans système de localisation. En d'autres termes, elle exprime l'augmentation en temps de retrait qui aurait été enregistrée en l'absence d'un système de localisation. Cette dernière correspond au gain en pourcentage du temps de retrait des requêtes localisées sur le temps de retrait des requêtes sans localisation.

$$\downarrow \%\ Temps = 100 - \frac{Temps\ de\ retrait\ des\ requetes\ avec\ localisation}{Temps\ de\ retrait\ des\ requetes\ mirorees\ sans\ localisation} * 100$$

6.3.3 Diminution du trafic réseau

Cette mesure exprime l'amélioration en nombre de paquets envoyés pour l'ensemble des requêtes mirorées par rapport à une configuration sans système de localisation. En d'autres termes, elle exprime l'augmentation en nombre de paquets envoyés sur le réseau qui aurait été enregistrée en l'absence d'un système de localisation. Cette dernière correspond au gain en pourcentage du nombre de paquets des requêtes localisées sur le nombre de paquets des requêtes sans localisation.

$$\downarrow \%\ Trafic = 100 - \frac{Nb\ de\ paquets\ envoyes\ par\ les\ requetes\ avec\ localisation}{Nb\ de\ paquets\ envoyes\ par\ les\ requetes\ mirorees\ sans\ localisation} * 100$$

6.3.4 Coût mémoire

Cette métrique exprime la somme des espaces mémoires utilisées par un système de localisation. Cette dernière est fonction du nombre d'entrées dans l'ensemble des bases de données utilisées par le système de localisation.

6.3.5 Coût de la coopération

Le coût de la coopération peut être mesuré par l'augmentation du trafic inter-sites coopérants et par l'augmentation de la charge due à la coopération.

Augmentation du trafic inter-sites coopérants

Cette métrique exprime la somme du nombre de messages échangés dans le cadre d'un système de localisation coopératif. Cette dernière est fonction du nombre d'entrées dans les différentes bases de données des sites coopérants.

Augmentation de la charge due à la coopération

D'un point de vue système, il faudrait évaluer s'il est profitable pour un organisme de coopérer avec d'autres organismes. Nous calculons l'augmentation de la charge d'un système de localisation due à la coopération comme étant l'augmentation du nombre de requêtes de localisation avec coopération par rapport au nombre de requêtes de localisation sans coopération.

6.4 Définition des scénarios

Possédant les traces et les candidats à mirorer (voir chapitre 5), il convient de trouver un placement pertinent à ces derniers par rapport aux utilisateurs. Pour cela, nous découpons notre espace Internet en zones comme décrit dans la section suivante. Ensuite, nous décrivons l'ensemble des scénarios choisis et les résultats obtenus en rejouant les traces.

6.4.1 Placement des miroirs : *Distance temporelle*

Il est important de bien placer les miroirs par rapport aux utilisateurs ciblés (que ce soit des miroirs érigés par le serveur d'origine ou par les utilisateurs eux mêmes). En premier temps, nous sommes tentés de traduire « bien placés » en terme de proximité géographique des miroirs par rapport aux utilisateurs (et qu'on exprimera probablement en nombre de routeurs) . Bien que cette mesure varie moins en fonction du temps [CC95, CC96a], elle ne garantit en rien l'objectif recherché, i.e la rapidité d'accès. Ceci est dû au fait que les liens sur Internet sont tellement hétérogènes que nous pourrions tout à fait avoir un site géographiquement proche auquel nous sommes connectés par des liens de faible bande passante (ou congestionnés), et ainsi avoir un accès plus lent qu'un site lointain mais très bien connecté. Dans notre étude, nous nous basons sur une définition temporelle de proximité que nous appellerons *distance temporelle*.

La *distance temporelle* permet d'exprimer la latence nécessaire pour récupérer des documents à partir d'un site *relativement* à un utilisateur (ou un organisme) . Nous proposons dans ce qui suit une classification *temporelle* des zones (voir fig. 6.1) copiée sur la classification géographique habituelle.

- **Zone locale** : c'est une zone de latence $t <= t_{local}$
- **Zone métropolitaine** : c'est une zone de latence $t_{local} < t <= t_{metropolitain}$
- **Zone nationale** : c'est une zone de latence $t_{mtropolitain} < t <= t_{national}$
- **Zone continentale** : c'est une zone de latence $t_{national} < t <= t_{continental}$
- **Zone intercontinentale** : c'est une zone de latence $t > t_{continental}$

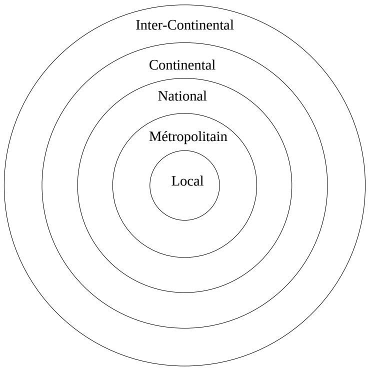

Fig. 6.1 – *Découpage en zones*

Ce découpage temporel sert pour le placement des miroirs. Ainsi, les sites classés dans une certaine zone ne seront mirorés que dans des zones inférieures en terme de latence. La définition des bornes temporelles est variable selon l'appréciation de l'utilisateur.

À titre d'exemple, un utilisateur accédant rapidement à un site aurait tendance à le déclarer dans la *zone locale* ou la *zone métropolitaine* alors qu'un autre qui accède au même site mais avec beaucoup de difficulté le classerait plutôt dans la *zone continentale* ou la *zone intercontinentale*.

6.4.2 Choix des *distances temporelles*

Dans ce qui suit, nous définissons notre vision de ce que pourraient être les temps d'accès vers les différentes zones. Nous avons voulu faire correspondre notre définition de *zone locale* à un réseau local assez rapide. Pour cela, nous avons entrepris des mesures de la latence d'accès sur notre réseau local en envoyant des requêtes « ICMP ECHO REQUEST » (ping, voir 6.5.2), de tailles égales à l'ensemble des différents documents de nos traces, ceci afin d'avoir une bonne estimation de la latence. Pour les autres définitions de temps de latence, nous avons estimé de nous même la qualité de service attendue dans chacune des zones. Le tableau 6.1 explicite les temps que nous avons définis.

Site local	$t_{local} <= 3\ ms$
Site métropolitain	$3\ ms < t_{mtropolitain} <= 30\ ms$
Site national	$30\ ms < t_{national} <= 100\ ms$
Site continental	$100\ ms < t_{continental} <= 300\ ms$
Site intercontinental	$t_{intercontinental} > 300\ ms$

TAB. 6.1 – *Définition des zones pour nos expérimentations*

6.4.3 Choix des scénarios

Les scénarios utilisés dans l'étude sont purement hypothétiques mais néanmoins réalistes dans le sens où les miroirs définis sont plausibles. En effet, si les traces collectées identifient un site populaire, il est fort probable que ce dernier le soit pour d'autres communautés ayant les mêmes centres d'intérêt et ainsi qu'il lui soit créé un ou plusieurs miroirs. Ce sont nos sites candidats à être mirorés déterminés au chapitre 5.

Les systèmes de localisation ne doivent répertorier que les miroirs dont les temps d'accès sont inférieurs à ceux du site d'origine. Ainsi, dans tous les scénarios définis, nous les avons placés dans des zones inférieures à celles de leurs serveurs d'origine correspondant. Ainsi, un serveur classé en *zone intercontinentale* aura ses miroirs placés en *zone continentale*, et ainsi de suite.

Les scénarios définis correspondent à une certaine qualité de service (QoS) exprimée en terme de latence maximale d'accès acceptable définie par l'utilisateur. Ainsi, pour chaque scénario, nous définissons une zone limite correspondant à une certaine QoS. Les sites candidats à être mirorés se trouvant à l'extérieur de cette zone se verront attribuer des miroirs dans la zone limite. Les zones limites des scénarios sont explicitées dans le tableau 6.2.

	Zone limite
Scénario 1	*zone locale*
Scénario 2	*zone métropolitaine*
Scénario 3	*zone nationale*
Scénario 4	*zone continentale*

TAB. 6.2 – *Scénarios rejoués*

À titre d'exemple, le scénario 4 correspond à la condition $latence <= t_{continental}$ pour tous les accès. Alors, tous les sites candidats se trouvant en *zone intercontinentale* auront leurs miroirs en *zone continentale* et les autres les auront en leurs zones respectives.

6.4.4 Caractéristiques des scénarios

Nb de sites candidats à être mirorés	Nb de sites candidats à être mirorés restant
1978	1431

TAB. 6.3 – *Miroirs restant après 1 an*

Dans le tableau 6.4, on décrit les caractéristiques des traces des scénarios. On remarque que le nombre de miroirs annoncé au chapitre 5, ainsi que le pourcentage de requêtes mirorées n'est pas le même. Ceci est dû au fait que les traces dataient de plus d'un an lorsque nous avons commencé à les traiter et ainsi, plusieurs sites avaient déjà disparus (voir tableau 6.3).

	Nb de requêtes	Nb de sites mirorés	% requêtes mirorées > Zone Limite
Scénario 1	6270564	1421	36.1
Scénario 2	6270564	1306	32.2
Scénario 3	6270564	1186	23.5
Scénario 4	6270564	837	15.7

TAB. 6.4 – *Caractéristiques des scénarios*

6.5 Résultats obtenus

Dans cette section, nous allons commencer par définir nos hypothèses de travail pour les différents scénarios définis. Puis, nous allons décrire les différentes mesures effectuées et ensuite donner l'ensemble des résultats obtenus.

6.5.1 Hypothèses

Dans ce qui suit, nous considérons que la découverte d'un miroir se passe de la façon suivante :

1. un utilisateur demande un document à un serveur
2. ce dernier lui retourne d'une façon ou d'une autre une liste de miroirs

Cette méthode mime ce qui se passe actuellement sur le Web pour la découverte de miroirs.

6.5.2 Mesure de la latence

Nous définissons la *latence* d'un message de taille `L` entre deux nœuds A et B comme étant le temps entre l'envoi de ce message de A et la réception complète de ce message en B. Cette latence sera utile pour le calcul du *gain en temps global* (voir 6.3.2).

Tout d'abord, la latence que nous cherchons à mesurer est une latence relative aux utilisateurs. Ainsi, il aurait fallu lancer nos mesures à partir de machines appartenant aux trois sites de nos traces. Or, n'ayant pas la possibilité d'accès aux machines de deux de ces sites (INT et ENST), nous nous sommes contenté de lancer nos mesures à partir de l'INRIA et d'utiliser les mêmes résultats pour ces trois sites.

Pour mesurer la latence, nous avons envoyé des requêtes « ICMP ECHO REQUEST » (`ping`) vers les sites candidats à être mirorés. La latence étant une fonction non linéaire de la taille du document demandé, nous avons dû envoyer à chacun des ces sites plusieurs requêtes `ping` avec des tailles de paquets égales à un sous ensemble des tailles des documents qui leurs sont demandés. Mais cela ne suffit pas puisque la latence dépend aussi énormément de l'état du réseau. Ainsi, nous avons répété l'étape précédente à longueur de journée de travail de nos utilisateurs (les utilisateurs de nos traces ; dans notre cas, nous avons fait des `ping` de 8h du matin jusqu'à 8h du soir à des intervalles de 4h). Un `ping` retournant la valeur du « Round Trip Time » (temps aller retour), le résultat est divisé par 2 pour exprimer la latence.

Au moment du traitement des traces, certains sites n'existaient plus alors que d'autres filtraient les requêtes du type `ping` pour des raisons de sécurité. Pour contourner partiellement ce problème, nous avons utilisé, dans le deuxième cas, les latences retournées par l'utilisation de la commande `traceroute` (voir la section 6.5.3).

6.5.3 Mesure du trafic réseau

Nous définissons le *nombre de paquets d'un message* comme étant le nombre de paquets nécessaires pour le transport du message d'un nœud A vers un nœud B. Le *nombre de paquets* sera utilisé dans le calcul de *Diminution du trafic réseau* (voir la section 6.3.3).

Le *nombre de paquets d'un message* dépend du nombre de nœuds traversés par le message et la fragmentation du message sur chaque lien. Le premier est calculé « approximativement » à l'aide de l'utilitaire `traceroute`. Le dernier est impossible à obtenir puisque nous ne possédons aucune information sur les liens traversés.

De ce fait, on s'est alors contenté de calculer plutôt le trafic en volume d'octets envoyés sur le réseau. Cette mesure correspond au volume du message multiplié par le nombre de liens traversés. Bien qu'elle ne soit pas très exacte, cette mesure nous donne une « petite » idée sur les différences d'utilisation de la bande passante entre les différents systèmes de localisation.

6.5.4 Calcul du coût mémoire

Le coût mémoire considéré est celui lié directement au système de localisation : la taille de l'ensemble des bases de données miroirs nécessaires. Dans le cadre des scénarios, la base de donnée de chaque système (sauf le coopératif) est construite en fonction du nombre de requêtes non résolues par le système de localisation (« miss »). En coopératif, la base de donnée est construite par les requêtes non résolues et les apports des autres organismes coopérants.

Concrètement, le coût mémoire des systèmes de localisation *personnels* et *par groupe* se réduit au nombre de « miss » global. Pour le système de localisation par *coopération*, le coût s'élève au nombre de « miss » multiplié par le nombre de sites coopérants. Pour le système idéal, il n'est autre que le nombre de sites mirorés.

6.5.5 Calcul du coût de la coopération

Le coût de la coopération est exprimé en nombre de messages échangés entre les sites coopérants ; c'est le *trafic inter-sites coopérants*. Ce nombre est majoré par le nombre de messages envoyés contenant une entrée à la fois. Il est calculé en multipliant le nombre d'entrées dans la base de données miroirs multiplié par $(N-1)$, N étant le nombre de sites coopérants.

Quant à la *charge due à la coopération*, elle est presque inexistante dans le cas d'$AR\tilde{A}$ puisque la seule augmentation de charge est due à la réception des messages des sites coopérants concernant la construction de la base de donnée.

6.5.6 Résultats

Localisation	↑% Hit	↓% latence	↓% trafic	Coût mémoire	Coût coopération max
personnel	97.80	92.44	65.33	49879	0
par groupe	99.85	96.30	68.44	3215	0
par coopération	99.93	96.63	68.72	$1421 * 3 = 4263$	$1421 * 2 = 2042$
optimal	100	96.79	68.82	1421	0

TAB. 6.5 – *Scénario 1 : Résultats*

Localisation	↑% Hit	↓% latence	↓% trafic	Coût mémoire	Coût coopération max
personnel	97.80	81.97	38.87	44469	0
par groupe	99.85	85.23	40.53	2932	0
par coopération	99.93	85.50	40.66	$1306 * 3 = 3918$	$1306 * 2 = 2612$
optimal	100	85.64	40.71	1306	0

TAB. 6.6 – *Scénario 2 : Résultats*

Localisation	↑% Hit	↓% latence	↓% trafic	Coût mémoire	Coût coopération max
personnel	97.43	71.84	40.51	37838	0
par groupe	99.82	74.43	42.28	2632	0
par coopération	99.92	74.61	42.42	$1186 * 3 = 3558$	$1186 * 2 = 2372$
optimal	100	74.73	42.46	1186	0

TAB. 6.7 – *Scénario 3 : Résultats*

6.6 Analyse des résultats

Dans cette section, nous répondons à travers les résultats exposés dans la section précédente aux interrogations concernant l'utilité de la localisation et le système de localisation le plus adapté.

6.6.1 Utilité de la localisation

Les résultats obtenus (voir tableaux 6.5, 6.6, 6.7, 6.8) montrent une nette amélioration de la qualité de service (latence et trafic réseau) par rapport à une configuration sans localisation. L'amélioration se chiffre entre 60% et 97% pour le temps de retrait des documents et entre 42% et 69% de moins en nombre de trafic réseau. Ce résultat était bien sûr prédictible du fait que l'accès à un site mieux placé par rapport aux utilisateurs ne peut engendrer que des effets positifs. Mais qu'en est-il du coût de cette localisation ?

Les deux coûts mesurés (6.5.4 et 6.5.5) pour les différents scénarios montrent que le coût à payer pour la localisation, par tous les systèmes étudiés, reste minime. Ceci est dû au petit nombre d'entrées dans la base de données.

Le tableau 6.9 montre une estimation sur la taille en mémoire de la base de données des miroirs. Cette estimation se fait en considérant que pour chaque entrée dans la base de donnée, on a 9 localisations possibles. En moyenne, la longueur d'un nom de site dans nos traces est ~ 17 octets. La formule d'estimation est alors : Coût $= N * 17 * 10$, N étant le nombre d'entrées dans la base de données.

Comme on peut le constater, les tailles de stockage sont peu considérables par rapport aux moyens de stockage de nos jours. L'espace mémoire utilisé globalement par les systèmes de localisation personnels reste le plus important et atteint presque 35 fois la taille des autres systèmes. Cependant, il ne faut pas oublier que cet espace de stockage est distribué sur l'ensemble des machines des utilisateurs et de ce fait majoré sur chaque machine par l'espace mémoire nécessaire pour garder l'ensemble des entrées de la base de donnée optimale.

Quant au coût de la coopération, il est inexistant, même étant majoré. En conclusion, les différents systèmes de localisation apportent une nette amélioration de la qualité de service avec un coût presque inexistant, d'où leur grande utilité.

Localisation	↑% Hit	↓% latence	↓% trafic	Coût mémoire	Coût coopération max
personnel	97.50	61.36	42.11	24805	0
par groupe	99.81	65.87	43.63	1842	0
par coopération	99.91	66.84	43.75	$837 * 3 = 2511$	$837 * 2 = 1674$
optimal	100	66.95	43.79	837	0

TAB. 6.8 – *Scénario 4 : Résultats*

	personnel	par groupe	par coopération	optimal
Scénario 1	8.1	0.52	0.69	0.23
Scénario 2	7.2	0.48	0.64	0.21
Scénario 3	6.13	0.43	0.58	0.19
Scénario 4	4.0	0.30	0.41	0.14

TAB. 6.9 – *Estimation du coût mémoire en Moctets*

6.6.2 Comparaison entre les systèmes de localisation

L'utilité étant établie, nous nous intéressons à identifier le système de localisation le plus adapté. Les résultats obtenus (voir tableaux 6.5, 6.6, 6.7 et 6.8) concernant l'augmentation du taux de hit, la diminution du temps de latence global et du trafic réseau montrent que l'amélioration de la qualité de service offerte en présence de n'importe lequel des systèmes de localisation étudiés est pratiquement la même : entre 0.07% et 2.57% de différence pour le taux de hit, entre 0.11% et 5.59% de différence pour la latence et entre 0.04% et 3.49% de différence pour le trafic réseau. De même, comme on l'a constaté dans la section précédente, les coûts mémoire sont presque pareils pour l'ensemble des systèmes, sauf dans le cas de la localisation personnelle où ils atteignent jusqu'à 35 fois l'espace utilisé par d'autres systèmes. Mais ce chiffre reste insignifiant par rapport aux tailles des espaces mémoire disponibles et de ce fait ne change en rien le fait que les systèmes de localisation mesurés ne se démarquent pas pour les scénarios utilisés. On remarque toutefois que le système de localisation par coopération est le plus performant (celui qui s'approche le plus du système optimal).

En fait, ces résultats s'expliquent par la faible différence d'entrées entre les systèmes de localisation (voir l'entrée *Coût mémoire* dans les tableaux 6.5, 6.6, 6.7 et 6.8). En effet, cette différence du nombre d'entrées correspond au nombre de requêtes non servies par un système de localisation par rapport à un autre (augmentation des « miss »), et de ce fait introduisent une chute de la qualité de service. Cette dernière reste très négligeable due au nombre restreint de ces cas.

Cette constatation pourrait induire que la coopération, choisie pour notre système $AR\tilde{A}$ n'est pas très intéressante. Or, il n'en est pas le cas, puisque ces résultats sont très dépendants des scénarios choisis, et surtout des types de miroirs choisis. Dans les scénarios précédents, on a estimé que les miroirs sont éventuellement découverts par tous les systèmes de localisation, ce qui est loin d'être le cas. Par exemple, pour les miroirs créés « à la volée » (qu'on appellera *miroirs marginaux*) pour combler un besoin ponctuel d'une certaine communauté, il faudrait un système de publication à la $AR\tilde{A}$ pour faire connaître ce nouveau miroir à l'ensemble des communautés intéressées. Et si on extrapolait en considérant que pour tous nos scénarios précédents, l'ensemble des miroirs sont marginaux, on aurait alors un gain non existant pour les systèmes de localisation sans mécanisme de publication. L'hypothèse de l'existence des miroirs marginaux est confortée par la quantité non négligeable ($\sim 28\%$) de sites non existants lors du traitement des traces après presque 1 an de la collecte.

En conclusion, la coopération dans les systèmes de localisation se démarque totalement dans des environnements dynamiques, avec des créations de miroirs marginaux pour des périodes de temps variables, où les autres systèmes de localisation restent inactifs et fournit une qualité de service supérieure à coût égal ou moindre dans les environnements classiques (où la découverte des sites est à la portée de tout le monde).

6.7 Conclusion

Dans ce chapitre, nous avons effectué des mesures sur des traces réelles afin de quantifier l'apport de différents systèmes de localisation et statuer sur leur utilité. Tout d'abord, nous avons identifié différentes métriques pour la comparaison de systèmes de localisation, en général. Ensuite, nous avons procédé à la définition des scénarios mimant la réalité. Les résultats obtenus montrent sans ambiguïté une nette amélioration en temps de retrait et une moindre utilisation du réseau pour la totalité des systèmes étudiés, les coûts étant négligeables. La localisation par coopération à la $AR\tilde{A}$ a donné les meilleurs résultats. Cependant, elle ne s'est pas démarquée énormément des autres. Nous avons pu constater, alors, que la différence en qualité de service avec les différents systèmes réside plutôt dans ses fonctionnalités de publication et découverte, qui s'avèrent indispensables dans des environnements mutants.

Quatrième partie

Conclusion

Chapitre 7

Conclusion

Sommaire

7.1 Introduction

Les accès aux serveurs Web sur des systèmes à large échelle souffrent de problèmes de performance et de disponibilité dus aux grandes distances géographiques entre les sites et à leur grand nombre. La réplication des serveurs sous forme de miroirs dans des sites proches de leurs utilisateurs permet d'augmenter la disponibilité et d'améliorer les performances.

Actuellement, la localisation des miroirs n'est pas transparente. Elle est centralisée par les serveurs sources et exige une participation active de l'utilisateur. En outre, la localisation des miroirs marginaux (non connus du serveur source) n'est pas possible.

L'objectif de ce travail était de proposer une solution à la localisation des miroirs pour le large échelle. Nous avons tout d'abord effectué des études sur des traces dans le but de mieux comprendre les accès Web et de proposer une solution appropriée. Ensuite, nous avons conçu et mis en œuvre $AR\tilde{A}$, un système de localisation de miroirs, qui utilise la coopération entre *organismes* et la communication épidémique et paresseuse pour offrir un support d'accès transparent aux miroirs les plus appropriés. Enfin, nous avons réalisé des mesures de performance sur des traces pour vérifier le gain attendu par l'utilisation de ce système.

7.2 Résultats

Nous résumons ici les principaux résultats de ce travail en trois axes : 1) la création d'un modèle de coopération adapté à la localisation des miroirs ; 2) la conception d'une architecture répartie qui passe à l'échelle ; 3) la définition d'une méthodologie d'évaluation. Les contributions de la thèse sont détaillées ensuite.

7.2.1 Coopération épidémique et paresseuse

Notre étude sur les traces nous a permis de constater que, parmi les serveurs accédés, moins de 10% étaient responsables de la plupart du volume de trafic. Ceci nous a permis de choisir un mécanisme spécifique utilisant le fait que le nombre des serveurs intéressants à mirorer est peu important. Dans ce sens, nous avons choisi de localiser les miroirs sans pour autant mettre en œuvre un mécanisme global qui localiserait tous les miroirs mais qui coûterait cher en ressources et en temps. La solution que nous avons retenue pour répondre à ces critères consiste à localiser uniquement les miroirs susceptibles d'être accédés. Ne les connaissant pas à l'avance, nous nous sommes fonde's sur la localité d'intérêts entre groupes d'utilisateurs pour anticiper un accès éventuel à un miroir.

La coopération entre groupes d'utilisateurs géographiquement proches les uns des autres nous a parue une solution plus adaptée à la réalité des accès Web et moins coûteuse.

Le passage à l'échelle du mécanisme de coopération est assuré par une communication épidémique et paresseuse des messages. L'échange épidémique permet à des sites qui n'ont aucun lien direct de s'échanger des informations. Il s'effectue au niveau des groupes d'*organismes*. Un *organisme* participant à deux groupes véhicule les messages dans les deux groupes. Ces messages contaminent ainsi les *organismes* membres de ces groupes même s'ils ne communiquent pas directement. Plusieurs groupes ainsi liés forment un graphe connexe d'échanges de messages de méta-données. Si ce graphe englobe un grand nombre des groupes d'*organismes*, l'échange épidémique permet de supporter le côté large échelle dans le système de communication. Le système de communication épidémique fonctionne avec un mécanisme d'échange de messages paresseux. Les messages sont envoyés par *piggybacking* ou par *batch*. En outre, ce mécanisme limite l'envoi de messages lorsque le système est chargé. Toutes ces mesures permettent le passage à l'échelle et diminuent le coût de la coopération.

7.2.2 Architecture répartie

Le système $AR\tilde{A}$ est construit sur une architecture répartie basée sur des *cellules* $AR\tilde{A}$. L'architecture répartie de $AR\tilde{A}$ est un compromis entre une répartition totale qui augmente le nombre de messages mais améliore les performances, et une centralisation poussée qui diminue le nombre de messages mais crée un goulot d'étranglement. La centralisation se fait au niveau de chaque *organisme*, qui collecte et conserve toutes les requêtes de ses utilisateurs. La répartition est utilisée pour la coopération inter-*organismes* et pour l'échange épidémique de messages inter-groupes d'*organismes*.

Deux bases de données sont utilisées pour mettre en œuvre le système : la base de données de miroirs (BDM) et la base de données de participants (BDP). La BDM est répliquée dans chaque organisme, et une *cohérence occasionnelle* à l'intérieur d'un groupe d'*organismes* coopérants permet de partager la connaissance des miroirs. La coopération épidémique inter-groupes d'*organismes* assure une mise à jour entre les BDMs. La BDP est répliquée avec une cohérence forte[12] pour chaque *organisme* d'un même groupe.

La coopération répartie dans $AR\tilde{A}$, associée à la centralisation des requêtes à l'intérieur d'une *cellule* permet de connaître rapidement la localisation d'un miroir. $AR\tilde{A}$ améliore la performance de localisation grâce à un coût local de redirection de la requête.

Le mécanisme de répartition de la BDM, associé à la centralisation des requêtes à l'intérieur d'une *cellule*, permet une connaissance locale généralisée des miroirs intéressant la communauté. Les performances de localisation sont donc améliorées grâce au coût local de la redirection des requêtes.

7.2.3 Méthodologie d'évaluation et mesures de performance

La méthodologie d'évaluation utilisée dans ce travail a été définie pour faire face aux difficultés d'évaluer et de valider les architectures à large échelle. En particulier, l'impossibilité de connaître les miroirs existants[13] a été décisif dans le choix de notre solution. Notre objectif était de comprendre l'intérêt de la coopération dans la localisation des miroirs et ensuite de mesurer le gain et le coût de l'utilisation de $AR\tilde{A}$.

Notre méthodologie est basée sur des études de traces réelles. Dans un premier temps, l'étude de ces traces nous a aidé à mieux comprendre les caractéristiques des accès Web. Ces ca-ractéristiques nous ont permis de conclure positivement sur l'intérêt des miroirs et sur la coopération comme solution à leur localisation. Ensuite, et dans le cadre de ces traces, nous avons identifié les serveurs responsables de la plupart du volume du trafic et une liste des candidats intéressants à mirorer a été définie.

Dans un deuxième temps, nous avons utilisé ces traces pour simuler un déploiement du système $AR\tilde{A}$ parmi trois *organismes* (INRIA, INT et ENST). Pour pouvoir réaliser cette simulation, nous avons défini le principe de la *distance temporelle* qui standardise une distance logique dans l'Internet. L'utilisation de la *distance temporelle* pour créer des scénarios pour le placement des miroirs nous a permis d'effectuer des mesures de performances sur l'utilisation de $AR\tilde{A}$. Les résultats de ces mesures sont encourageants. Néanmoins, nous nous attendons à des résultats encore plus significatifs lors d'un déploiement plus important de $AR\tilde{A}$.

7.2.4 Contributions

Les principales contributions de ce travail concernent les points suivants :

- l'intégration des principes d'échanges épidémiques des messages dans un modèle de coopération pour la localisation des miroirs ;
- la définition des principes tels que : la *distance temporelle* dans le cadre des mesures de performance pour le placement des miroirs et le *degré de contamination* dans la technique de filtrage des informations ;
- la description et l'utilisation d'une méthodologie pour l'étude des traces ;
- la conception et la réalisation du système $AR\tilde{A}$.

Chacun de ses points a été développé en considérant la base du problème original : la localisation des réplicats dans un environnement large échelle et le miroir comme forme de réplication.

12. Dans les *groupes ouverts* la cohérence forte n'est pas exigée
13. Principalement les miroirs marginaux (non connus du serveur source).

7.3 Perspectives

Même si le système actuel est fonctionnel et démontre l'intérêt de l'approche proposée, nous envisageons des travaux d'amélioration et d'extension.

Le premier point concerne le déploiement de $AR\tilde{A}$ dans plusieurs *organismes* et groupes d'*organismes*, de façon à préciser les mesures de performances.

Le deuxième point vise à améliorer le passage à l'échelle et optimiser la vitesse d'échange de messages, et pour cela, nous envisageons de hiérarchiser le graphe formé par les groupes d'*organismes*.

Le troisième point concerne l'affectation dynamique du *degré de contamination* par chaque groupe d'*organismes* qui devrait permettre un filtrage des informations plus fin et éviter une contamination trop large.

Enfin, nous envisageons d'affiner les stratégies de choix du miroir le plus approprié, dans le but de diminuer le coût de ces stratégies et de mieux correspondre aux demandes des utilisateurs.

7.3.1 Déploiement de $AR\tilde{A}$

Les mesures de performances actuelles ne sont relevées que sur des traces (réelles), et une évaluation du système sur des données en cours d'utilisation, nécessitant un déploiement du système sur plusieurs *organismes*, permettrait de confirmer nos résultats actuels. Plus généralement, une utilisation réelle, avec plusieurs *cellules* $AR\tilde{A}$ et quelques groupes d'*organismes* fournirait une évaluation significative et surtout complète du système, assurant notamment le passage à l'échelle.

7.3.2 Hiérarchisation de $AR\tilde{A}$

Les limites de $AR\tilde{A}$ portent particulièrement sur le nombre d'organismes qui peuvent coopérer et sur la taille de la BDM supportée par chaque organisme. Ces limites dépendent principalement de la quantité d'information à échanger entre les *organismes*, de la latence du réseau et des algorithmes de filtrage d'information.

Pour étendre les limites imposées par la régionalisation de la coopération, nous pouvons hiérarchiser le graphe crée par les groupes d'*organismes*, et instaurer des serveurs de messages à chaque nœud de la hiérarchie, chaque serveur représentant un ensemble de groupes d'*organismes*. L'objectif est de raccourcir le chemin de propagation d'un message ainsi que de diminuer le nombre des messages en permettant une politique de *pull* (envoi des informations par le serveur à la demande de l' *organisme* [14]) dans les serveurs $AR\tilde{A}$ hiérarchiquement au-dessus. Cette solution prévoit également un entrepôt étendu de méta-données sur la localisation des miroirs représentant les BDM des groupes $AR\tilde{A}$ subordonnés au même nœud hiérarchique.

Nous pensons que cette architecture hiérarchisée, avec des serveurs de messages et des entrepôts hiérarchiques communs, peut faciliter le filtrage des informations et diminuer la quantité d'information à échanger entre les *organismes*.

7.3.3 Filtrage de l'information

Nous envisageons une version automatique et dynamique de l'affectation du *degré de contamination* par les groupes d'*organismes*. L'intérêt de chaque groupe d'*organisme* peut varier selon plusieurs facteurs. Par exemple, si le volume des messages diminue ou augmente, ou si le nombre

14. C'est un peu l'idée dans les *newsgroups*.

d'*organismes* membres du groupe varie, ou encore si le besoin de sécurité devient plus ou moins contraignant, etc.

L'automatisation de l'affectation du *degré de contamination* peut être implémentée via un mécanisme de *monitoring* de ces paramètres. Ensuite, la définition de certains seuils va permettre de décider de la modification ou non de la valeur du *degré de contamination*.

7.3.4 Choix du miroir

Les stratégies de choix du miroir le plus approprié fournies par $AR\tilde{A}$ peuvent bénéficier de plusieurs améliorations au niveau des performances et de la précision.

Pour le choix du miroir le plus rapide, il faut impérativement tenir compte de la charge de chaque miroir et pas seulement de son temps d'accès. Plusieurs techniques de détection de la surcharge, qui visent à répartir les requêtes, existent dans la littérature [ASDZ00, PAB$^+$98] et pourraient être utilisées. Une solution, pour détecter la surcharge et décider de rediriger les nouvelles requêtes vers des miroirs plus éloignés mais moins chargés, serait de tracer le temps de réponse des serveurs en faisant une différence entre le temps total de la requête et le temps de transfert (le *ping* du volume de la requête).

En ce qui concerne les problèmes de versions, mettre en place un protocole de coopération entre miroirs permettrait à tout miroir non-marginal de connaître la version de ses copies. Par ailleurs, l'utilisation d'un tel protocole de coopération entre les miroirs ouvre de nouvelles possibilités pour améliorer le maintien de la cohérence entre les miroirs d'une même source.

7.4 Conclusion

Dans cette thèse, nous nous sommes intéressés au problème d'amélioration des temps d'accès des utilisateurs à leurs données, dans le cadre de l'Internet et des environnement à large échelle en général. Nous proposons une solution originale, qui intègre des principes comme la communication épidémique, la coopération régionale, et qui cherche à remédier à un problème reconnu de sous-utilisation des miroirs, essentiellement pour des raisons de difficulté de nommage, de publication et de localisation.

Notre système de localisation des miroirs sur le Web, baptisé $AR\tilde{A}$, fournit une gestion des aliases de miroirs ainsi qu'un mécanisme coopérant et régional de publication et de découverte des miroirs. Ces services sont disponibles à travers un support d'accès transparent aux miroirs et un protocole de coopération paresseuse et épidémique entre groupes d'*organismes* qui s'échangent des informations sur la localisation des miroirs.

Le modèle de localisation des miroirs présenté est une solution non globale mais qui supporte l'environnement large échelle. On entend par solution globale celle qui englobe tous les miroirs de toutes les sources du Web. Une solution de telle envergure se heurte à des problèmes de distribution pour le stockage des méta-données, des problèmes de recherche et de mise à jour de la base de données. Notre solution est non globale mais vise à être globale dans les besoins de chaque *organisme*. Les *organismes* qui ont des intérêts communs doivent s'intéresser aux mêmes miroirs. Une certaine localité dans la recherche des documents sur le Web justifie le choix de la coopération non globale et plus ciblé.

Cette solution, qui présente des avantages certains de passage à l'échelle, laisse néanmoins un certain nombre de possibilités pour continuer dans cette voie, notamment par la hiérarchisation des *organismes* coopérants, le filtrage des informations ou la problématique de stratégie de choix des miroirs.

Annexes

Annexe A

Interface d'accès à la BDM

Les méthodes d'accès à l'objet `DBMirrorID` (base de données de miroirs) sont les suivants :

int ***addMirror***(MirrorID mid)
int ***delMirror***(MirrorID mid)
int ***delSource***(String sup)
int ***updateMirrorVersion***(MirrorID mid, Date d)
int ***updateMirrorVersion***(String urlp, Date d)
int ***updateMirrorMetric***(String urlp, int m)
MirrorID ***findMirrorByPrefix***(String urlp)
MirrorID ***findBestMirrorByPrefix***(String urlp)
MirrorIDList ***findSource***(String urlp)
MirrorIDList ***lookup***(String urlp)

int *addMirror*(MirrorID mid)

La méthode *addMirror* permet l'ajout d'un objet miroir 'mid' à la base de données. Tout miroir doit être forcément lié à un préfixe d'url source, l'identificateur unique de l'ensemble des réplicats de cette sous-arborescence. Cette méthode retourne un entier : zéro si tout s'est bien passé et -1 sinon.

int *delMirror*(MirrorID mid)

La méthode *delMirror* permet d'effacer un miroir après sa disparition. L'objet miroir 'mid' est identifié par son préfixe d'url. Cette méthode retourne un entier : zéro si tout s'est bien passé et -1 sinon.

int *delSource*(String urlp)

La méthode *delSource* permet d'effacer une liste de miroir correspondant à la même entrée dans la BDM. Cette méthode sert à éliminer une source après sa disparition. L'entrée de la BDM est identifiée par un préfixe d'url, soit d'un de ses miroirs soit de la source elle même. Cette méthode retourne un entier : zéro si tout s'est bien passé et -1 sinon.

int *updateMirrorVersion*(MirrorID mid, Date d)

L'appel *updateMirrorVersion* permet le changement de l'information liée à la version du miroir. L'objet miroir 'mid' identifie le miroir à modifier. Si la date 'd' passée en paramètre est plus récente que la date dans la base de données, alors la date est modifiée. Cette méthode retourne un entier : zéro si tout s'est bien passé et -1 sinon.

int *updateMirrorVersion*(String urlp, Date d)

L'appel *updateMirrorVersion* permet le changement de l'information liée à la version du miroir. Le préfixe url du miroir 'urlp' identifie le miroir à modifier. Si la date 'd' passée en paramètre est plus récente que la date dans la base de données, alors la date est modifiée. Cette méthode retourne un entier : zéro si tout s'est bien passé et -1 sinon.

int *updateMirrorMetric*(String urlp, int m)

L'appel *updateMirrorVersion* permet le changement de l'information liée à la métrique du miroir. Le préfixe url du miroir 'urlp' identifie le miroir à modifier. La métrique, qui représente le meilleur miroir de la même source, entraîne la redéfinition de toutes les autres métriques de la même entrée. Cette méthode retourne un entier : zéro si tout s'est bien passé et -1 sinon.

MirrorID *findMirrorByPrefix*(String urlp)

La méthode *findMirrorByPrefix* permet de trouver un miroir avec une url passée en paramètre. Cette url peut correspondre au préfixe url de la source ou d'un miroir quelconque. Le miroir retourné est le premier de la liste si le préfixe d'url passé correspond à celui de la source ou le miroir lui même sinon. Cette méthode retourne un MirrorID si le miroir est trouvé, sinon elle retourne 'null'.

MirrorID *findBestMirrorByPrefix*(String urlp)

La méthode *findBestMirrorByPrefix* permet de trouver le meilleur miroir pour une url passée en paramètre. Cette url peut correspondre au préfixe url de la source ou d'un miroir quelconque. Le miroir retourné est celui qui a la meilleur métrique selon la définition choisie. Cette méthode retourne un MirrorID si le miroir est trouvé, sinon elle retourne 'null'.

MirrorIDList *findSource*(String urlp)

L'appel *findSource* permet de trouver tous les miroirs correspondants à une source par un préfixe d'url passé en paramètre. Ce préfixe d'url peut correspondre à celui de la source ou d'un miroir quelconque. Cette méthode retourne une liste des miroirs correspondant à la même entrée de la base de données. Si l'url n'est pas connu de la BDM, un 'null' est retourné.

MirrorIDList *lookup*(String urlp)

La méthode *lookup* permet la localisation dans la base de données d'un miroir contenant l'url connu de l'utilisateur. Cette méthode retourne une liste des miroirs correspondant à la même entrée de la base de données. Si l'url n'est pas connu de la BDM, un 'null' est retourné.

Annexe B

Interface d'accès à la BDP

Les méthodes d'accès à l'objet base de données de participants sont les suivants :

AraProxy ***get***(OrganismID oid)
void synchronized ***put***(OrganismID oid,AraProxy pxy)
void ***remove***(OrganismID oid)
OrganismID ***updateOrganism***(OrganismID oOld, OrganismID oNew)
AraGroupSet ***add***(AraGroup araGroup)
AraGroupSet ***remove***(AraGroup araGroup)

AraProxy *get*(OrganismID oid)

La méthode *get* récupère le point de communication établie entre cette cellule et l'organisme 'oid'. Cette méthode retourne la socket qui permet la communication avec cet organisme.

void synchronized *put*(OrganismID oid, AraProxy pxy)

La méthode *put* insère dans la base de données l'organisme 'oid' avec le canal de communication entre la cellule et cet organisme. Cette méthode ne retourne rien.

void *remove*(OrganismID oid)

La méthode *remove* retire de la base de données l'organisme 'oid'. Cette méthode ne retourne rien.

OrganismID *updateOrganism*(OrganismID oOld, OrganismID oNew)

La méthode *updateOrganism* permet de mettre à jour l'identificateur d'un organisme. La mise à jour peut être sur son nom et/ou sur son *proxy*. Cette méthode retourne le nouveau objet OrganismID.

AraGroupSet *add*(AraGroup araGroup)

La méthode *add* insère dans la base de données le groupe d'organismes araGroup. Cette méthode retourne la base de données avec le nouveau groupe.

AraGroupSet *remove*(AraGroup araGroup)

La méthode *remove* permet d'effacer un groupe d'organismes de la base de données. Cette méthode retourne la base de données sans l'ancien groupe.

Bibliographie

[Aka] Akamai. http ://www.akamai.com/.

[all] alltheweb. http ://www.alltheweb.com/.

[Alt] Altavista. http ://www.altavista.com/.

[And87] D. Anderson. The dash project : Issues in the design of a very large scale distributed operating system. " " 87/338, Berkeley (USA), Jan 1987.

[APS98] Yair Amir, Alec Peterson, and David Shaw. Seamlessly selecting the best copy from internet-wide replicated web servers. In *DISC'98 - 12th International Symposium on DIStributed Computing*, Andros, Greece, Sep 1998.

[ASDZ00] Mohit Aron, Darren Sanders, Peter Druschel, and Willy Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. In *USENIX 2000 Annual Technical Conference*, San Diego, CA, jun 2000.

[BAZ+97] Samrat Bhattacharjee, Mostafa H. Ammar, Ellen W. Zegura, Viren Shah, and Zongming Fei. Application-layer anycasting. In *INFOCOM'97*, 1997.

[BB99] Krishna Bharat and Andrei Broder. Mirror, mirror on the Web : A study of host pairs with replicated content. In *Proceedings of the 8th International WWW Conference*, Toronto, Canada, mai 1999. `http://www8.org/w8-papers/4c-server/mirror/mirror.html`.

[BC98] Georges Brun-Cottan. *Cohérence de données répliquées partagées par un groupe de processus coopérant à distance.* Thèse de doctorat, Université Paris 6, Pierre et Marie Curie, Paris (France), September 1998.

[BDM98] Vincent Bouthors, Olivier Dedieu, and Mesaac Makpangou. Recommendations sharing forum for internet communities. Url, presentation, Dyade, Bull, INRIA, 1998. `http://webtools.dyade.fr/pharos/`.

[Bir85] Kenneth P. Birman. Replication and fault-tolerance in the ISIS system. Orcas Island WA (USA), December 1985.

[BJS87] Kenneth P. Birman, Thomas A. Joseph, and Frank Schmuck. Isis documentation : Release 1. Technical Report 87-849, Department of Computer Science, Cornell University, Ithaca, New York (USA), July 1987.

[BL95] T. Berners-Lee. Hypertext transfer protocol http/1.0. " ", HTTP working Group Internet draft, October 1995.

[BLMM94] T. Berners-Lee, L. Masinter., and M. McCahill. Uniform resource locators (url), Dec 1994. http ://www.w3.org/Adressing/rfc1738.txt.

[BLNS82] Andrew D. Birrell, Roy Levin, Roger M. Needham, and Michael D. Schroeder. Grapevine : An exercise in distributed computing. In *Communications of the ACM*, volume 25, pages 260–274, avr 1982.

[BO99] Greg Barish and Katia Obraczka. World wide web caching : Trends and techiques. Survey, USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey,CA 90292, USA, 1999.

[BZG94] B. Bhargava, Y. Zhang, and S. Goel. A study of distributed transaction processing in an internetwork. " ", Purdue University, Indiana (USA), 1994.

[CC95] M. Crovella and R. Carter. Dynamic server selection in the internet, 1995.

[CC96a] R. Carter and M. Crovella. Dynamic server selection using bandwidth probing in wide-area networks, 1996.

[CC96b] Robert L. Carter and Mark E. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. Technical Report BU-CS-96-007, Boston University, (USA), March 1996.

[CDN+95] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical Internet object cache. Technical Report 95-611, Computer Science Department, University of Southern California, March 1995. `ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/HarvestCache.ps.Z%`.

[CM86] D.R. Cheriton and T.P. Mann. A decentralized naming facility. " " STAN-CS-86-1098, Stanford University, Department of Computer Science (USA), 1986.

[DC94] F. Dagorn and C.Gross. World wide web. Technical report, Université de Rennes 1, France, April 1994.

[DCYK83] Oppen Derek C. and Dalal Yogen K. The clearinghouse : a decentralized agent for locating named objects in a distributed environment. In *ACM Transcations on Office Information Systems*, volume 1, pages 230–253, July 1983.

[DGH+87] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Sixth Symposium on Principles of Distributed Computing*, pages 1–12, Vancouver, Canada, August 1987.

[Dor99] Neilze Dorta. ArÃ : Un système de gestion des miroirs sur internet. In *Proceedings of the Journée des Jeunes Chercheurs en Systèmes (JCS99)*, Rennes, France, June 1999.

[DPS+94] Alan Demers, Karin Petersen, Mike Spreitzer, Douglas Terry, Marvin Theimer, and Brent Welch. The bayou architecture : Support for data sharing among mobile users. In *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, December 1994. http ://www.parc.xerox.com/csl/projects/bayou/.

[Ebl98] Maria R. Ebling. *Translucent Cache Management for Mobile Computing.* PhD thesis, Carnegie Mellon University, March 1998. http ://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-coda.html, CMU-CS-98-116.

[EGLT76] K.P. Eswaran, J.N. Gray, R. Lorie, and I.L. Traiger. The notions of consistency and predicate locks in a database system. *Communications of the ACM*, 19(11) :624–633, Nov 1976.

[Fid91] C. Fidge. Logical time in distributed computing systems. *IEEE Computer*, pages 28–33, July 1991.

[GL91] Richard Golding and Darrell D. E. Long. Accessing replicated data in a large-scale distributed system. Technical Report UCSC-CRL-91-01, Computer Research Lab., U. of California, Santa Cruz CA (USA), January 1991.

[Goo] Google. http ://www.google.com/.

[HBLL99] Mor Harchol-Balter, Tom Leighton, and Daniel Lewin. Resource discovery in distributed networks. In *Proceedings of the 18th ACM - SIGACT/SIGOPS Symposium on Principles of Distributed Computing*, Atlanta, GA, May 1999.

[HHB96] Abdelsalam A. Helal, Abdelsalam A. Heddaya, and Bharat B. Bhargava. *Replication Techniques in Distributed Systems.* KAP (Kluwer Academic Publishers), 1996.

[How88] J.H. Howard. An overview of the andrew file system. In *Proceedings of the USENIX Winter Technical Conference*, Dallas, TX, USA, February 1988. http ://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-afs.html.

[HST96] Franz J. Hauck, Maarten van Steen, and Andrew S. Tanenbaum. A location service for worldwide distributed objects. In M. Muhlhauser, editor, *ECOOP'96, Workshop on Mobility and Replication*, Special Issues in Object-oriented Programming, pages 384–388, Linz, Austria, July 1996.

[Isi91] Isis Distributed Systems. *The Isis Distributed Toolkit, Version 3.0, User Reference Manuel.*, 1991.

[JPR+83] D. Stott Parker Jr., Gerald J. Popek, Gerard Rudisin, Allen Stoughton, Bruce J. Walker, Evelyn Walton, Johanna M. Chow, David A. Edwards, Stephen Kiser, and Charles S. Kline. Detection of mutual inconsistency in distributed systems. *TSE*, 9(3) :240–247, 1983.

[KRR99] Jussi Kangasharju, Keith W. Ross, and James W. Roberts. Locating copies of objects using the domain name system. In *Proceedings of the 4th International Web Caching Workshop*, San Diego, California, April 1999.

[KRR00] J. Kangasharju, K. W. Ross, and J. W. Roberts. Performance evaluation of redirection schemes in content distribution networks. In *5th Web Caching Workshop*, Lisbon, Portugal, May 22 - 24 2000.

[KS91] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. In *Thirteenth ACM Symposium on Operating Systems Principles*, volume 25, pages 213–225, Asilomar Conference Center, Pacific Grove, US, 1991. http ://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-coda.html.

[KS92] James Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transaction on Computer Systems*, 10(1) :3–25, February 1992. http ://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-coda.html.

[KS93a] P. Kumar and M. Satyanarayanan. Supporting application-specific resolution in an optimistically replicated file system. In *4th Workshop on Workstation Operating Systems (WWOS-IV)*, pages 66–70, Napa, CA, US, October 1993. http ://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-coda.html.

[KS93b] Puneet Kumar and M. Satyanarayanan. Log-based directory resolution in the Coda file system. In *Second International Conference on Parallel and Distributed Information Systems*, pages 202–213, 1993.

[KSB+99] David Karger, Alex Sherman, Andy Berkheimer, Bill Bogstad, Rizwan Dhanidina, Ken Iwamoto, Brian Kim anf Luke Matkins, and Yoav Yerushalmi. Web caching with consistent hashing. In *The Eighth International World Wide Web Conference*, Toronto, Canada, May 11-14 1999.

[Kum94] P. Kumar. *Mitigating the Effects of Optimistic Replication in a Distributed File System.* PhD thesis, Carnegie Mellon University, December 1994. http ://www.cs.cmu.edu/afs/cs.cmu.edu/project/coda/Web/docdir/pkumar-thesis.ps.Z, CMU-CS-94-215.

[Kum95] P. Kumar. Flexible and safe resolution of file conflicts. In *USENIX Winter 1995 Technical Conference*, New Orleans, LA, January 1995. http ://www.cs.cmu.edu/afs/cs.cmu.edu/project/coda/Web/docdir/usenix95.ps.Z.

[Lam78] L. Lamport. Times, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7) :558–65, July 1978.

[LCC+97] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, and Stephen Wolff. A brief history of the internet. In *Communications of the ACM*, fev 1997. http ://www.isoc.org/internet-history/.

[Lit91] Marc C. Little. *Object replication in a Distributed System.* PhD thesis, University of Newcastle upon Tyne Computing Laboratory, Newcastle upon Tyne, England., September 1991.

[LLG92] Rivka Ladin, Barbara Liskov, and Sanjay Ghemawat. Providing hogh availability using lazy replication. In *ACM Transcations on Computer Systems*, volume 10, pages 360–391, November 1992.

[Mat88] F. Mattern. Virtual time and global states in distributed systems. In *Workshop on Parallel and Distributed Algorithms*, Elsevier (Holland), October 1988.

[MDZ98] Andy Myers, Peter Dinda, and Hui Zhang. Performance characteristics of mirror servers on the internet. *School of Computer Science, Carnegie Mellon University*, June 1998.

[MGLNS94] Mesaac Makpangou, Yvon Gourhant, Jean-Pierre Le Narzul, and Marc Shapiro. Fragmented objects for distributed abstractions. In T. L. Casavant and M. Singhal, editors, *Readings in Distributed Computing Systems*, pages 170–186. IEEE Computer Society Press, July 1994.

[Moc87] Paul Mochapetris. Dns : The domain name system. RFC 1034, Network Working Group, ISI, November 1987.

[MPKD98] Mesaac Makpangou, Guillaume Pierre, Christian Khoury, and Neilze Dorta. Replicated directory service for weakly consistent replicated caches. Rapport de Recherche RR-3514, INRIA, April 1998. http://www-sor.inria.fr/publi/RDSWCDC_rr3514.html.

[MrB97] Mesaac Makpangou and Éric Bérenguier. Relais : un protocole de maintien de cohérence de caches web coopérants. *NoTeRe '97 colloquium*, 4–6 nov 1997.

[Nee93] R.M. Needaham. *Distributed Systems.* Number Chapter 12. Addison Wesley, 1993.

[Neu92] B.C. Neuman. *The Virtual System Model : A Scalable Approach to Organizing Large Systems.* PhD thesis, University of Washington (USA), 1992.

[PAB+98] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Willy Zwaenepoel Peter Druschel, and Erich Nahum. Locality-aware request distribution in cluster-based network servers. In *8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, San Jose, CA, oct 1998.

[Par94] Craig Partridge. *Gigabitbit Networking.* Addison-Wesley, 1994.

[Pie99] Guillaume Pierre. *Architecture et dimensionnement d'infrastructures de caches Web pour Intranets décentralisés.* PhD thesis, Université d'Evry-val d'Essonne, Evry (France), June 1999. http://www-sor.inria.fr/publi/pierre_thesis99.html.

[PMM93] C. Partridge, T. Mendez, and W. Milliken. Host anycasting service. Request for comments 1546, (USA), November 1993.

[PST+97] Karin Petersen, Mike Spreitzer, Douglas Terry, Marvin Theimer, and Alan Demers. Flexible update for weakly consistent replication. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*, pages 3–6, Saint-Malo, France, October 1997. ACM.

[PSTT95] Karin Petersen, Mike Spreitzer, Douglas Terry, and Marvin Theimer. Bayou : Replicated database services for world-wide applications. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP-15)*, pages 3–6, Copper Mountain Resort, Colorado, December 1995. ACM.

[Pul98] Giray Pultar. Automatically selecting a close mirror based on network topology. In *12th Systems Administration Conference (LISA '98)*, Boston, Massachusetts, USA, Dec 6-11 1998.

[RGK95] M. Rabinovich, N. Gehani, and A. Kononov. Scalable update propagation in epidemic replicated databases, 1995.

[Sat89] M. Satyanarayanan. Coda : A highly available file system for a distributed workstation environment. In *Second IEEE Workshop on Workstation Operating Systems*, Pacific Grove, CA, USA, September 1989. http ://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-coda.html.

[Sat90] M. Satyanarayanan. Scalable, secure, and highly available distributed file access. *IEEE Computer*, 23(5), 1990. http ://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-coda.html.

[SEKN92] Michael F. Schwartz, Alan Emtage, Brewster Kahle, and B. Clifford Neuman. A comparison of internet ressource discovery approaches. *Computing Systems Journal, The USENIX Association*, 4(5), August 1992.

[SHT97] Maarten van Steen, Philip Homburg, and Andrew S. Tanenbaum. The architectural design of globe : A wide-area distributed system. Technical Report 422, vrije Universiteit - Faculty of Mathematics and Computer Science, Amsterdam - Netherlands, March 1997.

[SHTH98] Maarten van Steen, Philip Homburg, Andrew S. Tanenbaum, and Franz J. Hauck. Locating objects in wide-area systems. *Communications Magazine*, pages 2–7, jan 1998.

[SKK+90] M. Satyanarayanan, J. J. Kistler, P. Kumar, M.E. Okasaki, E.H. Siegel, and D.C. Steere. Coda : A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4) :447–459, 1990. http ://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-coda.html.

[SN79] M. Stonebraker and E. Neuhold. Concurrency control and consistency of multiple copies of data in distributed ingres. *IEEE Transactions on Software Engineering*, 3(3) :188–194, May 1979.

[Sou94] Khaled Saadi Soufi. *Quorum consensus protocols for the client-server architecture.* PhD thesis, Department of Computer Science University of Illinois at Urbana-Champaign, Urbana, Illinois, 1994.

[Sta95] W. Stallings. *ISDN and Broadband ISDN with frame relay and ATM.* Prentice Hall, New Jersey, 1995.

[Tac97] Chantal Taconet. *Graphes de Réseaux Coopérants et Localisation Dynamique pour les Systèmes Répartis sur Réseaux Étendus.* PhD thesis, Université d'Évry Val d'Essone, France., October 1997.

[TB96] C. Taconet and G. Bernard. Object location in wide area networks for microkernel-based operating systems. In *ECOOP '96 Workshop on Mobility and Replication*, Linz, Austria, Jul 8 1996.

[TB97] C. Taconet and G. Bernard. Dynamic object location in wide area networks. In *2nd European Research Seminar on Advances in Distributed Systems (ERSADS)*, pages 17–21, Zinal, Switzerland, Mar 1997.

[TDP+94] Douglas Terry, Alan Demers, Karin Petersen, Mike Spreitzer, Marvin Theimer, and Brent Welch. Session guarantees for weakly consistent replicated data. In *International Conference on Parallel and Distributed Inormation Systems*, Austin, TX, US, September 1994. http ://www.parc.xerox.com/csl/projects/bayou/.

[TDVK98] Renu Tewari, Michael Dahlin, Harrick M. Vin, and Jonathan S. Kay. Beyond hierarchies : Design considerations for distributed caching on the internet. Technical Report TR98-04, Department of Computer Science, The University of Texas at Austin, Austin, Texas 78712-1188 (USA), 1998.

[Ter87] Douglas B. Terry. Caching hints in distributed systems. *IEEE Transactions on Software Engineering*, 13(1) :48–54, 1987.

[Tho79] R.H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Databse Systems*, 4(2) :180–209, Jun 1979.

[TTP+95] Douglas Terry, Marvin Theimer, Karin Petersen, Alan Demers, Mike Spreitzer, and Carl H. Hauser. Managing update conflict in bayou, a weakly connected replicated storage system. In *15th ACM Symposium on Operating Systems Principles*, Copper Mountain Resort, Colorado, US, December 1995. http ://www.parc.xerox.com/csl/projects/bayou/.

[URL] URL. Naming and adressing : Urls. http ://www.w3.org/Adressing/.

[Voi] Voilà. http ://www.voila.fr/.

[VtYB97] P. Vixie, S. thomson, Y.Rekhter, and J. Bound. Dynamic updates in the domain name system (dns update). RFC 2136, Network Working Group, ISI, April 1997.

[Wei85] P. Weiss. Yellow pages protocol specification. Technical report, Sun Microsystems Inc., August 1985.

[Yah] Yahoo. http ://www.yahoo.com/.