Spécialisation de serveurs par des agents mobiles

L. Ismail, D. Hagimont

Projet SIRAC (IMAG-INRIA)
INRIA, 655 av. de l'Europe, 38330 Montbonnot Saint-Martin, France

Internet: {Leila.Ismail, Daniel.Hagimont}@imag.fr

Résumé:

La recherche en systèmes répartis a vu récemment l'émergence d'un nouveau modèle pour la structuration d'applications réparties sur Internet : la programmation par agents mobiles. Dans ce modèle, un agent est un processus possédant un contexte d'exécution, incluant du code et des données, pouvant se déplacer de machine en machine (appelées serveurs) afin de réaliser la tâche qui lui est assignée.

Un des principaux avantages des agents mobiles est de permettre de spécialiser les serveurs d'information disponibles sur l'Internet en fonction des besoins des clients de ces serveurs. Il est en effet difficile pour un serveur de fournir une interface ou un service satisfaisant l'ensemble de ses clients potentiels. Les agents mobiles peuvent être utilisés afin de télécharger une extension sur les serveurs et d'ajuster ainsi les services fournis aux besoins spécifiques des clients.

Dans cet article, nous présentons cette classe d'applications des agents mobiles et rapportons notre expérience qui a consisté à développer et évaluer trois exemples d'applications. Ces trois exemples ont été développés sur un système à agents mobiles minimal que nous avons réalisé sur Java. Nous montrons que des gains d'efficacité significatifs peuvent être obtenus en étendant des serveurs pour des besoins spécifiques.

1. Introduction

L'informatique répartie actuelle est principalement fondée sur le modèle client-serveur. Les exemples les plus connus sont probablement le World Wide Web et les systèmes de type Corba [OMG91]. Cependant, deux problèmes importants restent difficiles à traiter :

- efficacité. Le coût des communications entre les clients et les serveurs reste élevé, ce qui explique les efforts des chercheurs pour optimiser les services de communication et pour développer des techniques de cache visant à masquer les lenteurs des réseaux.

- généricité. Il est difficile pour un serveur de fournir un service satisfaisant tous ses clients potentiels. De plus, si un serveur vise à fournir un service plus générique, cela se traduit souvent par une interface plus complexe impliquant des interactions plus fréquentes entre les clients et le serveur, d'où une perte en terme d'efficacité pour les clients.

Récemment, la recherche en systèmes répartis a vu l'émergence d'un nouveau modèle pour la structuration d'applications réparties sur Internet : la programmation par agents mobiles [KGR96]. Dans ce modèle, un agent est un processus possédant un contexte d'exécution, incluant du code et des données, pouvant se déplacer de machine en machine (appelées serveurs) afin de réaliser la tâche qui lui est assignée. Généralement, un agent réalise des appels de méthode sur des objets exportés par les serveurs visités ou par des agents rencontrés sur ces serveurs.

Les agents mobiles permettent de répondre aux deux problèmes mentionnés ci-dessus [CHK94]. Un serveur peut fournir un service et une interface très simples et génériques à ses clients. Cette généricité permet aux clients ayant des besoins spécifiques d'adapter le service. Afin de limiter le coût induit par les nombreuses interactions entre le client et le serveur, le client peut programmer un agent mobile qui est envoyé et exécuté sur la machine du serveur. Cet agent mobile implante une extension du serveur qui réalise exactement le service désiré par le client.

Dans cet article, nous décrivons notre expérience d'utilisation d'une plate-forme à agents mobiles pour la spécialisation de serveurs. Nous avons développé trois applications démontrant l'intérêt des agents mobiles pour cette classe d'applications. Ces applications ont été respectivement implantées avec un outil d'appel à distance classique, puis avec des agents mobiles spécialisant les serveurs. L'évaluation montre que des gains significatifs peuvent être obtenus grâce aux agents mobiles.

L'article est structuré de la façon suivante. Dans la section 2, nous présentons brièvement ce qu'est un système à agents mobiles et nous décrivons le système utilisé. La section 3 présente comment les agents mobiles peuvent être utilisés afin de spécialiser des serveurs d'information. La section 4 décrit les trois applications que nous avons développées afin de démontrer l'intérêt de cette approche, puis nous rapportons les résultats de notre évaluation de ces applications en section 5. Après une comparaison aux travaux similaires en section 6, nous concluons en section 7.

2. Environnement d'agent mobile

De nombreux systèmes à agents mobiles ont été développés ces dernières années. Des exemples sont les systèmes AgentTcl [GCKR96], Telescript [White94], Aglets [Venners97] ou MOA [MLC98]. Une grande partie de ces systèmes sont basés sur l'environnement

Java [GM95], principalement pour sa forte diffusion dans le monde, mais aussi ses avantages techniques : masquage de l'hétérogénéité des machines et typage fort pour la sécurité.

Nous rappelons tout d'abord les fonctions générales des systèmes à agents mobiles, puis nous décrivons la plate-forme que nous avons utilisée. Il s'agit d'une plate-forme minimale que nous avons implantée sur Java.

2.1. Fonctions des agents mobiles

Un agent est un processus pouvant se déplacer de machine en machine afin de réaliser une tâche. En général, la mobilité est fournie par le biais d'une primitive *move (machine*) qui permet de se déplacer vers la machine désignée par le paramètre.

Un agent est composé de son code correspondant à un algorithme, ainsi que d'un contexte incluant des données. Ce contexte peut évoluer en cours d'exécution, par exemple en collectant des données lorsqu'un agent réalise une recherche d'information sur un ensemble de serveurs. Le code et le contexte de l'agent sont déplacés avec l'agent lorsque celui-ci visite différents serveurs.

Lorsqu'un agent se déplace vers un serveur, il doit poursuivre son exécution sur le site destination. La plupart des systèmes à agents mobiles implantent une migration faible, c'est à dire une fonction de migration où l'agent redémarre son exécution depuis le début. En conséquence, le programmeur doit inclure dans le contexte de l'agent des informations sur l'état de l'exécution, et lorsque l'agent redémarre sur un site, le code de l'agent doit vérifier l'état de l'exécution et se brancher sur la partie de l'algorithme devant être exécutée sur ce site.

Un système à agent fournit en général des primitives de communication permettant aux agents d'interagir, mais aussi aux agents d'interagir avec les serveurs qu'ils visitent. Ces primitives de communication prennent la forme d'envois de messages ou d'appels de procédures ou de méthodes.

2.2. Notre plate-forme sur Java

Afin de mener notre expérimentation, nous avons réalisé une plate-forme à agents minimale sur l'environnement Java. Cette plate-forme minimale nous a permis d'identifier de façon fine les mécanismes mis en œuvre dans la gestion d'agents mobiles. Son principal atout est sa simplicité, ce qui nous a permis de l'installer sur des sites très éloignés dans le monde afin de réaliser une évaluation de cette technologie dans des conditions proches de la réalité.

Dans cette plate-forme, un agent est une classe Java héritant de la classe *Agent*. La classe de l'agent doit définir une méthode *main ()* qui est le point d'entrée de l'agent. A chaque arrivée sur un nouveau site, cette méthode *main* () est appelée sur l'objet agent qui vient d'être transféré sur le site. La classe *Agent* fournit deux méthodes : *move (machine)* permettant de se déplacer sur le site *machine* et *back ()* qui permet à l'agent de revenir à son site d'origine.

Lorsqu'un agent est déplacé, un message est construit, contenant le code de l'agent ainsi que le contexte de l'agent. Le contexte de l'agent est constitué d'un ensemble d'objets Java. Le mécanisme de sérialisation de Java [RWWB96] nous permet de transformer l'ensemble des objets gérés par l'agent (un graphe d'objets) en un tableau d'octets, de déplacer ce tableau vers le site destination, puis de reconstruire ce graphe. Le mécanisme de chargeur de code de Java (*ClassLoader*) nous permet de transporter le code de l'agent vers le site destination, puis de créer dynamiquement les classes Java associées sur ce site. Dans la plate-forme minimale, un agent n'est composé que d'une seule classe. Un agent ne doit donc contenir que des références à des objets qui sont des instances de classes définies par l'environnement Java (dont les classes sont présentes sur tous les serveurs visités). Cette limitation actuelle ne nous a pas gêné dans la mise en œuvre des exemples d'application décrits dans les sections suivantes.

Dans notre plate-forme, un agent communique avec les serveurs par appel de méthode Java. Lorsqu'un agent arrive sur un site, il a la possibilité de consulter un serveur de noms propre au site et qui permet de récupérer des références Java d'objets locaux à ce site. Le serveur peut donc exporter des références dans ce serveur de nom, permettant aux agents d'utiliser un ensemble d'objets Java gérés par le serveur.

3. Spécialisation de serveurs

Dans cet article, nous montrons comment les agents mobiles peuvent être utilisés pour adapter aux besoins des clients un service fourni par un serveur sur Internet. Nous supposons que l'objectif du serveur est de fournir un service générique pouvant répondre aux besoins très divers de tous ses clients potentiels.

Afin de fournir un service générique, le serveur peut exporter une interface très paramètrable, mais la complexité de cette interface augmente avec la diversité des utilisations possible du service. De plus, il est difficile d'identifier les besoins potentiels de tous les clients. Une autre approche consiste à décomposer le service en un ensemble d'opérations élémentaires, plus simples, pouvant être appelées indépendamment les unes des autres. Un client peut alors combiner les appels à ces opérations élémentaires afin d'obtenir le service désiré.

La deuxième approche est préférable, car elle permet d'offrir un service générique sans forcément connaître à l'avance les besoins des clients. Par contre, elle nécessite des interactions plus fréquentes entre le client et le serveur. Dans le cadre d'une architecture répartie fondée sur le modèle client-serveur, les interactions coûtent cher et la généricité devient alors très pénalisante. En utilisant des agents mobiles, un client peut développer un service spécialisé en fonction de ses besoins en s'appuyant sur le service générique fournit par le serveur. Le service spécialisé du client peut être envoyé sur le site du serveur sous la forme d'un agent mobile, ce qui implique que les interactions entre le service spécialisé et le service générique seront des appels locaux sur la machine serveur. La figure 1 illustre ce principe.

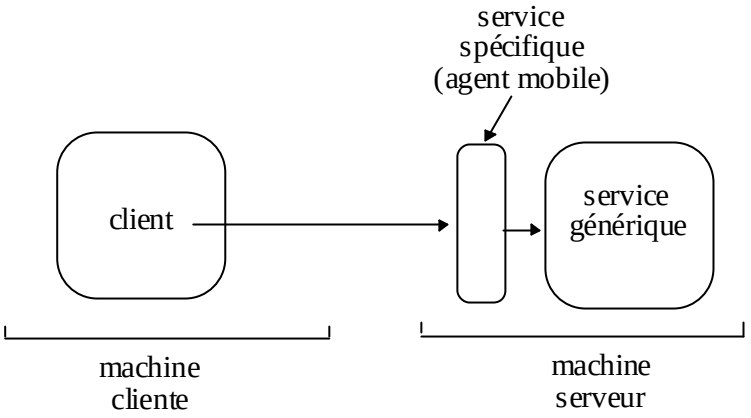

Figure 1. Extension d'un serveur par un agent mobile.

Ce principe, qui consiste à co-localiser une partie du code d'une application cliente et le service générique qu'elle utilise, afin de rendre plus efficace les interactions entre ces deux entités, n'est pas nouveau. C'est ce principe qui a motivé les travaux autour des extensions de systèmes d'exploitation (Spin[BSPS95], Exokernel [RKJ95]). Un service système générique nécessite de nombreux appels systèmes qui sont coûteux. Les systèmes extensibles visent donc à faire s'exécuter une partie de l'application dans le même espace d'adressage que le système d'exploitation, afin de réduire le coût des appels au système par cette partie de l'application. Nous nous proposons d'appliquer ce principe entre utilisant des agents mobiles dans le cadre d'applications réparties utilisant des serveurs d'information.

Un aspect important de ce principe est que l'envoi du service spécifique sur la machine serveur peut se faire suivant deux modes :

- installation préalable. Le service spécifique est installé sous la forme d'un agent par le client sur le site serveur. Le client peut ensuite réaliser des appels à distance au serveur en utilisant une interface spécifique exportée par l'agent qu'il a installé sur le serveur. L'installation de l'agent n'a lieu qu'une fois et toutes les utilisations du service sont ensuite réalisées par appel à distance.
- migration systématique. L'application du client est un agent mobile qui se déplace vers le serveur afin de réaliser des appels locaux sur le site serveur. A chaque appel au service, l'agent client est déplacé vers le site du serveur.

Avec une installation préalable, on ne paie le coût du déplacement d'un agent que lors de l'installation et les appels au service sont sans surcoût et bénéficient de l'adaptation du service. Avec la migration systématique, on paie le coût du déplacement d'un agent à chaque appel. Il faut noter que l'installation préalable d'une spécialisation peut être proscrite par le serveur si celui-ci ne désire pas voir s'accumuler des extensions du service sur son site.

Dans la suite de cet article, nous supposons une migration systématique d'agent, car notre objectif est d'évaluer l'intérêt d'une spécialisation d'un service à l'aide d'un agent mobile,

en fonction du coût de déplacement d'un agent. Dans le cas d'une installation préalable et une fois cette installation réalisée, le gain est évident.

4. Exemples de spécialisation de serveur

Afin d'évaluer l'intérêt des agents mobiles pour la spécialisation de serveurs, nous avons développé trois exemples d'applications que nous décrivons maintenant.

4.1. Redirection de requête

Une des applications les plus importantes dans le domaine des agents mobiles est la recherche d'information. Dans ces applications, des agents se déplacent sur différents sites pour chercher des informations pour leurs clients.

Notre exemple est une application répartie sur Internet dont le but est de chercher une liste d'hôtel dans une ville. Dans notre scénario, deux bases de données doivent être consultées. La première recense des hôtels et permet d'obtenir la liste des hôtels de la ville où le client désire se rendre. La deuxième base de données est un annuaire permettant d'obtenir les numéros de téléphone de ces hôtels. Ces deux bases de données sont gérées sur des sites différents par des administrations ou des entreprises différentes (par exemple un office de tourisme et une compagnie de téléphone pour notre scénario).

Ces deux serveurs fournissent chacun une interface correspondant au service qu'ils fournissent. Pour le premier serveur, nous supposons que l'interface permet au client de donner le nom de la ville, le serveur lui retournant une table des hôtels dans cette ville. Pour le deuxième serveur, nous supposons que l'interface permet au client de donner le nom d'un hôtel, le serveur retournant le numéro de téléphone de cet hôtel.

En utilisant le modèle client-serveur classique, le client va devoir faire un appel à distance pour interroger le premier serveur et récupérer la table des hôtels qui l'intéressent. A partir de cette table, le client va ensuite, pour chaque hôtel dans la table, réaliser un appel à distance au second serveur afin de récupérer le numéro de téléphone de hôtel.

Ainsi, dans le mode client-serveur, le client doit appeler le second serveur (B) autant de fois qu'il y a d'éléments dans le tableau d'hôtels retourné par le premier serveur (A). Le temps d'obtention de la réponse finale est égal au coût de communication avec le serveur A (rpc(A)), plus le coût des communications avec le serveur B qui est égal au coût de l'appel au serveur B multiplié par la taille du tableau obtenu comme réponse du serveur A (n*rpc(B)). D'où un coût total[1] :

*coût total = rpc(A) + n*rpc(B)*

[1] Nous ne prenons pas en compte le cout de traitement d'une requête sur chaque serveur qui reste le même dans tous les cas de figure.

Les deux serveurs A et B fournissent chacun une interface générique composée d'opérations élémentaires. Les clients peuvent ainsi utiliser le service selon leurs besoins. Cependant, cela se traduit par de nombreuses interactions entre les clients et le serveur B.

Une solution bien plus efficace pour obtenir le résultat final est que le serveur A transmette directement la table des hôtels obtenue au serveur B afin que celui-ci réalise une jointure avec sa table des adresses. Pour ce faire, il faut spécialiser les serveurs pour qu'ils offrent ce service de redirection de requêtes. Une extension statique des serveurs (par les administrateurs des deux serveurs) n'est pas une solution réaliste dans la mesure où ces bases de données sont administrées séparément. De plus, il n'est pas réaliste d'étendre un serveur pour chaque besoin spécifique d'un client.

Nous proposons donc d'utiliser une spécialisation dynamique des serveurs par des agents mobiles (Figure 2).

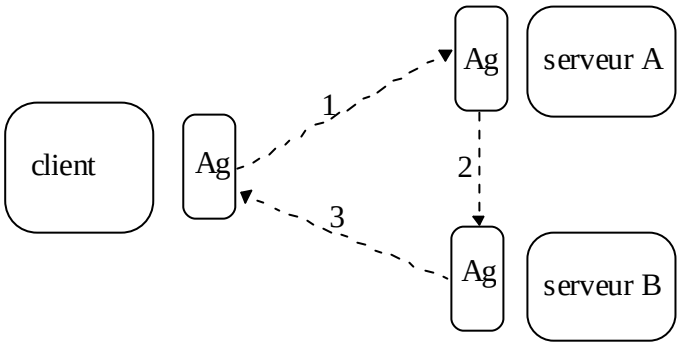

Figure 2. Déplacement de l'agent pour la redirection de requêtes

Le client crée un agent *Ag* contenant la requête globale à réaliser. Cet agent se déplace (1) tout d'abord sur le site du serveur A et réalise localement l'appel au serveur A pour obtenir la table des hôtels. La table des hôtels est stockée dans le contexte de l'agent, puis l'agent se déplace (2) vers le serveur B. Sur le serveur B, l'agent peut itérer sur la table des hôtels et demander au serveur B le numéro de téléphone de chaque hôtel (cette étape correspond à la jointure). Ces numéros de téléphones sont stockés dans le contexte de l'agent qui se déplace (3) finalement vers son site d'origine où il délivre le résultat au client.

Comme on le voit à travers cet exemple, l'utilisation d'agents mobiles permet d'étendre dynamiquement l'interface des serveurs. Les agents ont été utilisés ici pour permettre une redirection de requête entre des bases de données différentes.

Dans cette mise en œuvre avec des agents mobiles, un agent avec peu de données dans son contexte doit être envoyé sur le serveur A (coût *A*), puis l'agent se déplace avec la table des hôtels vers le serveur B (coût *A(Thotel)*), puis cet agent retourne à son site d'origine avec la table enrichie avec les numéros de téléphone (coût *A(Ttel)*). D'où un coût total de :

coût total = A + A(Thotel) + A(Ttel)

En conséquence, la solution à base d'agents mobiles sera plus efficace si :

*A + A(Thotel) + A(Ttel) < rpc(A) + n*rpc(B)*

Dans le scénario que nous considérons, nous montrons en section 5 à partir de quelle valeur de *n* la solution à base d'agents est rentable.

4.2. Compression de données

Le Web est très souvent utilisé pour récupérer des documents de toute sorte. Nous considérons dans cette section l'accès à un serveur exportant une interface permettant à un client d'obtenir une copie d'un document. Le client demande au serveur un document de nom donné et récupère un flot de données correspondant au contenu du document.

Cependant, si le client désire optimiser ce transfert de données en utilisant un outil de compression des données, il est nécessaire d'étendre le serveur pour y ajouter cette fonction de compression. De plus, différents clients peuvent utiliser différents outils de compression, nécessitant des extensions particulières pour chaque client.

Le paradigme d'agents mobiles offre une réponse élégante à ce problème. En effet, le client peut envoyer un agent sur le site du serveur. Cet agent interroge le serveur pour obtenir le document, compresse le document obtenu et revient sur le site du client avec le document compressé. Sur le site client, l'agent peut décompresser le document et le délivrer au client. De cette manière, le client réalise une extension dynamique du serveur en fonction de ses besoins spécifiques sans changer l'interface de ce dernier.

La principale motivation du client pour cette extension du serveur est l'efficacité du transfert du document. Pour étudier le coût de ce transfert, considérons deux machines dont le débit de transfert entre elles est *dr*. Nous supposons que le facteur de compression du document est *fc* (si *T* est la taille initiale du document, le document compressé a une taille de *fc*T*) et que les temps respectifs de compression et décompression sont *tc* et *td*. Nous cherchons à identifier les conditions à partir desquelles la compression du document est rentable.

Le temps de transfert du document sans compression est *T/dr* si *T* est la taille du document à transférer.

Le temps de transfert du document avec compression est :

*tc + fc*T/dr + td*

respectivement le temps de compression du document, le temps de transfert du document compressŽ et le temps de dŽcompression du document.

En conséquence, la compression est rentable si :

*tc + fc*T/dr + td < T/dr*

Ceci implique que le client gagne dans les cas où le débit du réseau entre sa machine et la machine du serveur est tel que :

$dr < T(1 - fc) / (tc + td)$

Cette formule indique à partir de quand la compression est rentable. Nous ne prenons pas en compte ici le coût du transfert de l'agent. Notre évaluation en section 5 montre que ce coût est largement amorti pour des documents de grande taille.

4.3. Interrogation de serveur en mode déconnecté

Une autre fonction à laquelle nous nous sommes intéressés est de permettre aux clients d'avoir des traitements en cours sur des serveurs pendant que les ordinateurs des clients sont déconnectés [GKNRC96]. Normalement, les serveurs exportent à leurs clients des interfaces devant être utilisées de façon synchrone. Ainsi, un client appelant un serveur reste bloqué en attente du résultat. Il ne peut pas envoyer une requête à un serveur, puis se déconnecter et récupérer la réponse à sa requête lors de sa prochaine connexion.

Le paradigme d'agent mobile permet aux clients de spécialiser le serveur en envoyant un agent vers le site du serveur, cet agent représentant le client sur le site serveur pendant la durée de la déconnexion du client. L'agent réalise l'appel synchrone au serveur, et il offre une interface permettant au client (par envoi de message) de lui signaler sa reconnexion pour provoquer le retour de l'agent avec la réponse à la requête.

Cette application qui illustre l'intérêt des agents mobiles pour la spécialisation de serveur a été réalisée sur notre plate-forme, mais ne donne pas lieu à une évaluation quantitative dans la section qui suit.

5. Evaluation

Dans cette section, nous présentons une évaluation des possibilités de spécialisation de serveurs en utilisant des agents mobiles, à partir des deux premières applications présentées ci-dessus : redirection de requêtes et compression de données. Nous avons développé ces applications avec et sans la spécialisation par des agents mobiles. Pour chacune des deux applications, nous avons réalisé une version en utilisant RMI (*Remote Method Invocation*) [WRW96], le mécanisme d'appel de méthode à distance de Java et en utilisant notre plate forme d'agents mobiles.

Les mesures ont été effectuées sur deux machines SunOS sparc et un PC Pentium sous Windows NT. Ces trois machines sont connectées à Internet et se trouvent dans trois unités administratives différentes réparties en France : une machine SunOS à l'INRIA-Paris (Paris), une machine SunOS à l'université de Grenoble (Campus) et le PC à l'INRIA-Grenoble (Grenoble). Nous avons programmé nos applications en utilisant la version jdk1.1.5 de la

machine virtuelle Java. Pour avoir une idée de la capacité de transfert entre ces machines, nous avons calculé les débits respectifs entre les machines, comme le montre la tableau 1.

Connexion	Débit (ko/sec)
Grenoble - Paris	134
Paris - Campus	83
Campus - Grenoble	130

Tableau 1. débit entre les machines

Nous avons également mesuré le coût de transfert d'un agent de taille minimale entre deux sites. Entre les sites Grenoble et Paris, l'aller-retour d'un tel agent coûte 375 millisecondes. Notre plate-forme n'a pas été optimisée, mais cette mesure donne un ordre de grandeur du coût de transfert d'un agent.

5.1. Redirection de requête

L'application de recherche des hôtels décrite en section 4.1 est géographiquement répartie comme suit :

- Paris : le site du serveur qui gère les hôtels
- Campus : le site du serveur qui gère l'annuaire
- Grenoble : le site du client

La réponse finale de l'application est un tableau constitué d'un ensemble d'enregistrements. Comme l'analyse de coût présentée en section 4.1 l'a montré, le temps de réponse de la requête varie avec le nombre d'enregistrements retournés.

Nous avons effectué des mesures en faisant varier le nombre d'enregistrements retourné par le premier serveur (retournant la table des hôtels). Dans notre application, la taille d'un enregistrement est de 80 octets. Le tableau 2 donne, pour différents nombres d'enregistrements, les coûts des exécutions basées sur RMI et sur un agent mobile réalisant la spécialisation présentée auparavant.

Nombre d'enregistrements	RMI (temps en millisecondes)	Agent (temps en millisecondes)
1	150	4200
200	7000	4200
400	14500	4200
600	19200	5000
800	25600	5500
1000	32700	6000
1200	39700	6500
1400	46700	7000

Tableau 2. Résultats obtenus pour la redirection de requêtes

On remarque que pour un nombre d'enregistrements petit (infŽrieur ˆ 200 enregistrements), RMI est plus efficace que la solution à base d'agents. Ceci s'explique par le fait que, pour un nombre d'enregistrements faible, le nombre d'appels à distance économisés par l'utilisation d'un agent est également faible et ne suffit pas à amortir le coût de la migration de l'agent. Cependant, avec un nombre d'enregistrements suffisant, la solution à base d'agents devient plus efficace. Ces mesures sont représentées sur la figure 3, montrant que la solution à base d'agents est d'autant plus intéressante que les interactions avec les serveurs distants sont nombreuses.

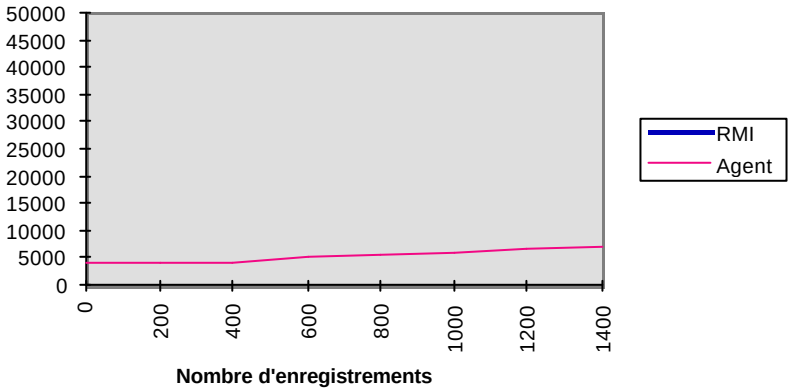

Figure 3. Comparaison de RMI et des agents mobiles pour la redirection de requête

5.2. Compression de données

Dans cet exemple, nous utilisons un agent mobile pour spécialiser un serveur de documents et mettre en œuvre une compression des documents avant transfert sur le réseau et leur décompression avant de les délivrer aux clients.

Nous avons calculé en section 4.3 que la compression de document devenait rentable lorsque $dr < T(1 - fc) / (tc + td)$, dr étant le débit du réseau, T la taille du document transféré, fc le facteur de compression du document et tc et td les temps respectifs de compression et décompression.

Pour réaliser notre évaluation, nous avons utilisé les sites de Grenoble et Paris présentés ci-dessus. Le document que nous avons utilisé est un fichier Postscript de taille 1,2 Mo. Nous avons utilisé l'outil de compression *gzip*[2] et avons obtenu avec notre fichier un facteur de compression de 30%. Les temps de compression et décompression mesurés sont 2,8 secondes et 0,2 secondes. En reportant ces valeurs dans la formule ci-dessus, nous obtenons que la compression devient rentable lorsque $dr < 280$ *Ko/s*. Etant données les débits mesurés entre les machines utilisées (présentés dans le tableau 1), la compression de documents doit être rentable dans notre environnement d'expérimentation sur Internet[3].

Cependant, le calcul précédent ne prend pas en compte le coût induit par l'utilisation d'agents mobiles pour spécialiser le serveur et lui faire faire de la compression de données. Les mesures données dans le tableau 3 correspondent à la recherche du document respectivement avec RMI sans compression et avec un agent mobile réalisant la compression (sur le site serveur) et la décompression (sur le site client).

RMI (temps en millisecondes)	Agent (temps en millisecondes)
9500	6200

Tableau 3. Coût de la recherche du document

Ces résultats montrent que le coût de migration de l'agent est largement amorti par le gain du à la compression du document.

[2] accessible à travers une classe Java.

[3] A noter que sur notre réseau local, nous obtenons un débit de l'ordre de 700 Ko/s pour lequel la compression n'est pas rentable.

6. Travaux similaires

Comme nous l'avons déjà mentionné en section 3, l'idée de permettre une extension d'un service en rapprochant une partie de l'application du service, afin de rendre plus efficaces les interactions entre ces deux derniers, n'est pas nouvelle. Elle a été notamment exploitée dans le cadre de projets de recherche autour des systèmes extensibles, comme par exemple dans les projets Spin[BSPS95] et Exokernel [RKJ95]. Ces systèmes permettent aux applications de spécialiser les services du système d'exploitation en incluant dans l'espace d'adressage du système un module de l'application. Ce module étend le système d'exploitation de la machine en fonction de besoins spécifiques de l'application. Dans notre cas, nous nous sommes intéressés à l'extension des services d'un serveur en déplaçant un module du client vers le site du serveur par le biais d'un agent mobile.

Cette approche, qui consiste à déplacer du code de la machine cliente vers la machine serveur afin d'exécuter un code spécifique au client sur la machine serveur, est aussi très utilisée dans le domaine des bases de données relationnelles réparties. Un serveur de gestion de base de données permet d'exécuter une requête en code SQL provenant de l'application d'un client. Cette requête est déplacée vers le site serveur et interprétée par le serveur. Cependant, même si cette architecture permet l'envoi d'une requête spécifique à un serveur générique, le jeu d'instruction du serveur se limite aux fonctions de gestion de la base de donnée, et surtout, il ne permet pas à un agent de se déplacer de sites en sites.

Antonio et al.[CPV97] ont étudié les trafics réseaux générés par des applications réparties construites en utilisant les paradigmes client-serveur et agents mobiles. Cette étude s'appuie sur un modèle de coût prenant essentiellement en compte les volumes d'information transférés. Nos travaux sont basés sur des expérimentations dans des conditions réelles d'utilisation des agents mobiles entre des machines réparties sur Internet. Nous avons montré l'intérêt de la spécialisation de services par des agents mobiles en développant trois applications. Nous avons évalué et comparé des versions de ces applications développées avec les technologies client-serveur (RMI) et agents mobiles.

7. Conclusion et perspectives

Dans cet article, nous nous sommes intéressés à l'utilisation d'agents mobiles pour la spécialisation de serveurs d'information. Au lieu d'ajouter une nouvelle interface ainsi qu'un service adapté pour chaque nouveau besoin différent d'un client, un serveur peut fournir un service générique que chaque client pourra utiliser en fonction de ses besoins spécifiques. Un service générique nécessite souvent des interactions plus nombreuses entre les clients et le serveur. Afin de limiter le coût de ces interactions, le serveur peut fournir un environnement d'exécution permettant à des agents mobiles de venir s'exécuter sur son site. Le client qui utilise le service générique en fonction de besoins précis se déplace sur le site serveur et réalise les appels au service localement.

Afin d'évaluer l'intérêt de cette approche, nous avons réalisé une plate-forme d'agents mobiles minimale sur l'environnement Java. Cette plate-forme fournit les fonctions de base généralement fournies par les systèmes à agents mobiles. Elle a été utilisée pour mettre en œuvre trois scénarii d'applications, visant à montrer l'intérêt d'une spécialisation du serveur par un agent mobile. Pour deux de ces applications, les mesures ont également montré que l'on peut obtenir des gains d'efficacité significatifs par rapport à une solution fondée sur le modèle client-serveur classique.

Le travail présenté dans cet article se poursuit à l'heure actuelle. La perspective la plus immédiate est de réaliser une évaluation plus fine en étudiant tous les mécanismes élémentaires mis en œuvre lors du déplacement d'un agent et leurs coûts respectifs. Par la suite, nous nous proposons d'étendre ces travaux pour permettre à une application de choisir à l'exécution la stratégie à utiliser. Nous avons montré que la spécialisation de serveur pouvait être rentable sous certaines conditions, en particulier avec des débits faibles et des interactions fréquentes entre le client et le serveur. Nous nous proposons d'étudier comment un client peut prendre la décision d'utiliser une extension du serveur par un agent mobile ou d'utiliser simplement l'approche client-serveur.

Bibliographie

[CHK94] D. Chess, C. Harrison et A. Kershenbaum, "Mobile Agents: are they a good idea ?", IBM Research Report, RC 19887, Décembre 1994.

[GCKR96] R. Gray, G. Cybenko, D. Kotz, and D. Rus: “Agent TCL”, Department of Computer Science, Dartmouth College, Hanover, NH, Mai 1996.

http://www.cs.dartmouth.edu/?agent/papers/chapter.ps.Z.

[GKNRC96] R. Gray, D Kotz, S. Nog, D. Rus et J. Cybenko, "Mobile Agents for Mobile Computing", Technical Report PCS-TR96-285, Department of Computer Science, Dartmouth College, Hanover, NH, Mai 1996.

[GM95] J. Gosling and H. McGilton, The Java Language Environment: a White Paper, Sun Microsystems Inc., 1995.

http://java.sun.com/whitePaper/java-whitepaper-1.html,

[KGR96] D. Kotz, R. Gray, and D. Rus : “Transportable Agents Support Worldwide Applications”, Department of Computer Science, Dartmouth College, Hanover, NH, Mars 1996.

[MLC98] D. S. Milojicic, W. LaForge, D. Chauhan, "Mobile Objects and Agents (MOA)", 4th USENIX Conference on Object-Oriented Technologies and Systems, Avril 1998.

[OMG91] OMG. The Common Object Request Broker: Architecture and Specification, OMG Document Number 91.12.1, Revision 1.1, Décembre 1991.

[RWWB96] R. Riggs, J. Waldo, A. Wollrath, K. Bharat, "Pickling State in the Java System", Computing Systems, 9(4), 1996.

[Venners97] B. Venners, "Under the Hood: The Architecture of aglets", Mars 1997.

http://www.trl.ibm.co.jp/aglets/

[White94] J.E. White, "Telescript Technology: The Foundation for the Electronic Market-place", General Magic Inc., Mountain View, CA, 1994.

[WRW96] A. Wollrath, R. Riggs, J. Waldo, "A Distributed Object Model for the Java System", Computing Systems, 9(4), pp. 291-312, 1996.