SONDe: Contrôle de densité auto-organisante de fonctions réseaux pair à pair

Erwan Le Merrer, Anne-Marie Kermarrec, Didier Neveux

SONDe: Contrôle de densité auto-organisante de fonctions réseaux pair à pair

E. Le Merrer[†], A.-M. Kermarrec [‡]et D. Neveux [§]

Longtemps dominés par les systèmes de partage de fichiers, les systèmes pair à pair s'ouvrent désormais à un large éventail d'applications telles que l'email, le DNS, la téléphonie, ou les caches répartis. Le bon fonctionnement de ces applications passe par l'utilisation de fonctions de base dont l'accès peut devenir un goulet d'étranglement si elle ne sont pas suffisamment répliquées dans le système. Dans cet article nous présentons SONDe, un algorithme qui permet une réplication automatique et adaptative de ces fonctions dans un réseau à très large échelle. Cet algorithme permet également de borner le nombre de sauts réseaux à effectuer entre un pair et une fonction, rendant ainsi prévisibles et paramétrables les latences attendues. Ceci est rendu possible grâce à une simple prise de décision basée sur l'étude du voisinage de chaque pair du réseau.

Keywords: auto-organisation, densité, latences, répartition de charge

1 Introduction

Alors que le paradigme du pair à pair se développe en réponse aux questions suscitées par les besoins de robustesse et de passage à l'échelle, le problème des latences dans ce type de réseaux est toujours sous les feux de la rampe. Outre les méthodes proposant une prise en compte de la proximité géographique [LM03, KPS05, MC02], les tables de hachage distribuées [AR01, IS01] s'attèlent à minimiser le nombre de sauts logiques nécessaires pour acheminer une requête vers le pair cible. Bien que reconnu comme une avancée majeure dans le domaine, les algorithmes de base des DHT ne nous semblent pas s'adapter au mieux à certains cas précis, comme par exemple l'accès à des fonctions répliquées[¶] ; en effet, la recherche d'un pair responsable du nom de la fonction visée conduit toujours à une même petite partie de l'espace d'adressage de la DHT. Ceci provoque un phénomène de déséquilibre de la charge globale de cette DHT [BG04], ainsi qu'une surcharge des pairs responsables en cas de fonction utilisée par l'ensemble des participants du réseau.

Nous proposons dans cet article SONDe, un algorithme simple et totalement distribué permettant de répartir de façon homogène des fonctions au sein d'un réseau de pairs. Cette répartition permet d'assurer qu'un pair peut accéder à une fonction en un nombre de sauts réseau borné. Nous présentons également une heuristique permettant de rendre la répartition des fonctions auto-adaptative à la charge d'utilisation du service proposé. L'algorithme ne repose que sur une analyse du voisinage de chaque pair et permet ainsi le passage à l'échelle.

2 SONDe

2.1 Modèle de système

Nous considérons un système de type pair à pair non structuré ou les pairs participent au protocole en transférant les messages vers leurs destinataires. La métrique utilisée dans ce papier, le *hop*, représente un saut logique entre deux pairs *voisins* de la couche réseaux de pairs. Un pair peut être un *pair standard* ou une *fonction*. Ce dernier représente un pair qui a acquis la qualité de fonction, et propose le service à son

[†]France Telecom R&D / IRISA,erwan.lemerrer@francetelecom.com
[‡]IRISA/INRIA, Anne-Marie.Kermarrec@irisa.fr
[§]France Telecom R&D, didier.neveux@francetelecom.com

[¶] Dans notre cas précis, une fonction à pour but de rendre un service aux pairs du réseau. Au regard de l'échelle réseau considérée, cette fonctionnalité doit être répliquée pour pouvoir satisfaire l'ensemble des requêtes

voisinage ; nous ne traiterons pas de cette ”acquisition” qui peut se faire par exemple par téléchargement d'un module sur un serveur, auprès des autres fonctions, ou plus simplement par activation d'un module déjà présent au sein de chaque pair.

2.2 Algorithme de réplication simple

L'idée de base de l'algorithme est de s'assurer qu'un pair peut, à tout moment, trouver un pair hébergeant une fonction dans son voisinage à au plus *nbHopsMax* hops de lui même.

Algorithme Chaque pair du réseau, standard ou fonction, effectue régulièrement (après un tirage d'une période d'attente aléatoire par exemple) une vérification à *nbHopsMax* hops pour déterminer la présence ou non d'une fonction (*function_check*) ; cette action est, de par la taille des réseaux visés, asynchrone.

- Dans le cas où le pair effectuant cette procédure est un pair standard :
  - il acquiert la qualité de fonction si la recherche est sans succès
  - si une fonction est trouvée, alors elle devient le fournisseur de service de ce pair
- Si c'est une fonction qui effectue le *function_check* :
  - aucune action ne se produit si aucune autre fonction n'est trouvée dans le voisinage à *nbHopsMax* hops de celle-ci
  - dans le cas contraire, elle perd sa qualité de fonction pour redevenir un pair standard

Dans cette partie nous prenons pour hypothèse que les pairs hébergeant une fonction peuvent toujours rendre le service désiré aux pairs qui lui sont rattachés, sans notion de saturation. Ainsi un pair possédant la qualité de fonction utilise naturellement le service présent en son sein, et les autres pairs standards se connectent à la fonction à moins de *nbHopsMax* qui minimise le plus le temps de réponse.

Evaluation Nous avons simulé un réseau de pairs non structuré, constitué de liaisons bidirectionnelles entre les pairs. Le réseau formé est un espace à 2 dimensions où les pairs sont connectés à un nombre limité de leurs voisins physiques. Pour les résultats présentés dans ce papier, le nombre de pairs considéré est de 10,000, le nombre moyen de voisins par pair est de 5 (de 4 à 7 voisins). Pour simuler l'asynchronisme des *function_check*, et donc le fait que plusieurs pairs peuvent prendre de façon concurrente la décision d'en effectuer un, 5% des pairs l'effectuent en même temps.

Afin de vérifier expérimentalement si l'algorithme converge vers un état stable (ici plus de création ni de suppression de fonctions), nous lançons le simulateur sur un réseau statique (pas de départs/arrivées de pairs) vierge de fonctions. Nous constatons que deux ou trois *rounds*[‖] suffisent pour que le réseau de pairs se stabilise. Un état stable signifie que la condition de distance maximale *nbHopsMax* d'un pair à sa fonction est satisfaite pour l'ensemble des pairs du système, et qu'aucune fonction n'apparaît dans la vue d'une autre fonction. La Figure 2 confirme ceci ; elle souligne aussi que dans près de 90% des cas, un pair peut trouver une fonction avant la borne maximale attendue.

La Figure 1 (a) confirme visuellement que SONDe permet une répartition homogène des fonctions à travers la couche pair à pair.

2.3 Algorithme de réplication adaptatif à la charge

Dans la section précédente, nous avons considéré que les fonctions ont la capacité de servir toutes les requêtes. Nous ajoutons maintenant une heuristique pour supporter la surcharge des fonctions. Cette surcharge peut intervenir lorsque les clients connectés à une fonction consomment trop de ressources en même temps, ce qui interdit l'acceptation de nouvelles requêtes de connexion provenant d'autres pairs du voisinage. Une solution est ici d'augmenter le nombre de fonctions dans les zones surchargées.

Algorithme A la valeur *nbHopsMax* (borne maximale) utilisée dans l'algorithme précédent est associée une variable *nbHops*, qui indique la distance courante jusqu'à laquelle le pair doit trouver une fonction. *nbHops* varie selon les conditions que nous exposons dans la suite de ce paragraphe. Deux variables *seuilSousCharge* et *seuilSurcharge*($0 < seuilSousCharge < seuilSurCharge < 100\%$) sont ajoutées aux fonctions, qui sont à même de mesurer leur paramêtre de charge. Elles permettent respectivement de définir

[‖] Un round est la période nécessaire à tous les noeuds pour effectuer leur *function_check*

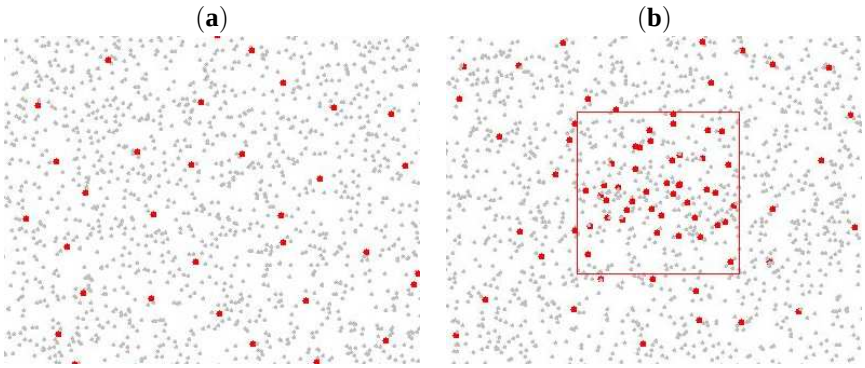

FIG. 1: (a) Répartition des fonctions dans le réseau de pairs à 2 dimensions (b) Conséquences d'une surcharge locale de fonctions

le taux de charge à partir duquel un mécanisme de réplication de fonctions doit être lancé, et en deçà duquel le nombre de fonctions doit être réduit pour libérer des ressources.

- Si le taux de charge reste pendant un quantum de temps prédéfini au dessus du seuil *seuilSurcharge*, alors la fonction concernée déclenche le processus suivant, qui conduit à la création de fonctions supplémentaires : elle notifie simplement à ses voisins (à $nbHops$) de modifier leur valeur $nbHops$ de la façon suivante : $nbHops = nbHopsFonction - 1$[**], et en fait de même ensuite pour sa propre valeur. $nbHops = 0$ signifie que le pair devient par conséquent fonction. Ce changement de profondeur de recherche à pour conséquence la création de nouvelles fonctions dans les zones où les pairs ne peuvent plus satisfaire la condition de distance maximale avec leur fonction.
- Dans le cas de figure inverse où la fonction reste sous chargée pendant le laps de temps prédéfini, alors elle modifie sa valeur ($nbHops = nbHops + 1$) et notifie ses voisins du changement de leur valeur de profondeur à $nbHopsFonction + 1$. Cette action provoque naturellement la suppression de fonctions alentours.

Afin de maintenir la borne supérieure désirée entre un pair et sa fonction, $nbHops$ ne peux excéder $nbHopsMax$; cette contrainte permet également la convergence de l'algorithme vers un état stable (i.e. identique à celui de l'algorithme à profondeur fixe) quand le réseau devient peu ou plus utilisé. Dû aux profondeurs variables auxquelles sont envoyées les notifications de changement de profondeur de recherche, certains pairs ne sont régulièrement pas atteints par celles-ci, et stagnent à des valeurs de $nbHops$ qui ne sont plus représentatives de la charge du réseau. Pour prévenir ceci, au bout d'un quantum prédéfini de rounds durant lesquels le pair n'a pas effectué de changement de sa variable, celui-ci va changer son $nbHops$ pour prendre comme valeur la moyenne de celle de ses voisins.

A la différence de l'algorithme à profondeur fixe, la suppression de fonctionnalité n'intervient que lorsqu'une fonction rencontre une autre fonction possédant la même profondeur de recherche, et ceci afin de conserver les modifications apportées.

Evaluation Pour les expérimentations présentées dans cet article, les seuils de sous-charge et de surcharge ont été fixés respectivement à 10% et 70% de la capacité de rendu de service simultané d'une fonction (maximum 10 pairs clients utilisant le service en même temps). L'évolution de la charge d'utilisation du service réseau (nombre de pairs utilisant ce service) provoque naturellement l'adaptation du nombre de fonctions : Figure 3 (a), tout en maintenant le service disponible : Figure 3 (b). Une des propriétés de cet algorithme est également de maintenir la localisation au sens logique des événements, c'est à dire de ne pas avoir d'impact global sur le réseau en cas de disparition/surcharge de fonctions, ou de liens par exemple. La Figure 1 (b) illustre ceci : une surcharge localisée d'une partie du réseau de pair, pour une couche pair à pair mappée sur le réseau physique, n'a d'influence que sur le voisinage immédiat des pairs concernés. Dans le cas où il n'existe pas de correspondance entre le réseau physique et le réseau logique, les liens étant

[**] Nous avons pu constater grâce aux expérimentations menées que le fait de décrémenter chaque valeur locale des pairs entraîne d'importantes fluctuations du nombre de fonctions créées, contrairement au fait d'imposer le passage au $nbHops$ de la fonction notificatrice, décrémenté d'une unité

tirés aléatoirement, ce cas de figure n'a pas lieu d'être.

SONDe se comporte bien face à la dynamique du réseau, avec à tout moment moins de quelques pourcents de pairs qui ne trouve pas une fonction à la distance désirée (Figure 2 (b)).

(**a**) (**b**)

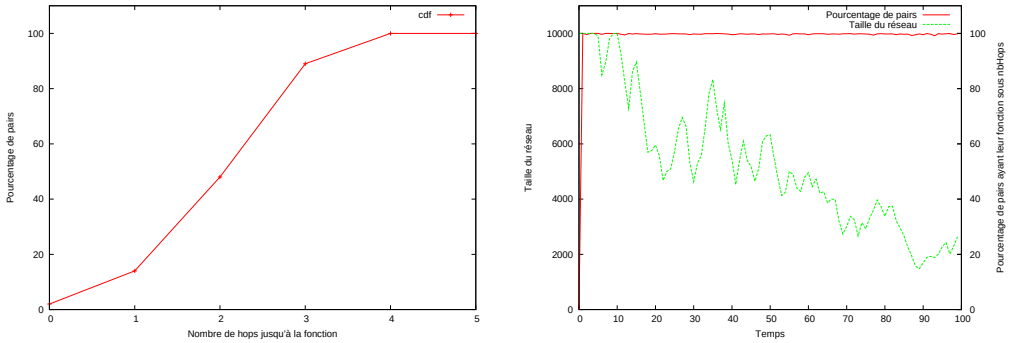

FIG. 2: (a) CDF de la distance entre les pairs standards et leur fonction attitrée (b) Evolution du pourcentage de pairs qui satisfont leur borne *nbHops*, sur un réseau avec départs/arrivées continus de pairs

(**a**) (**b**)

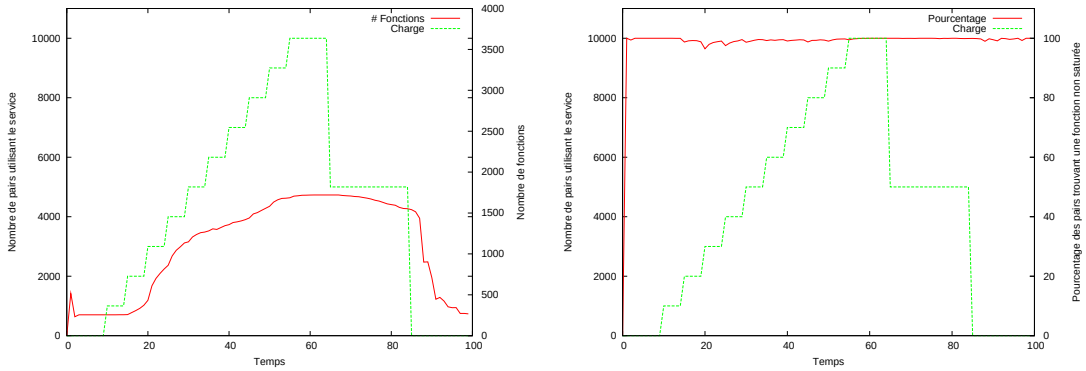

FIG. 3: Réaction face à la variation du nombre de pairs utilisant le service : évolution du nombre de fonctions (a) et du pourcentage (b) de pairs possédant leur fonction à *nbHops*. 10,000 pairs, service rendu par une fonction à au maximum 10 pairs simultanément

3 Conclusion et perspectives

Nous avons présenté SONDe, un algorithme simple de répartition et de duplication adaptatif de fonctions s'appuyant sur une prise de décision locale des pairs du réseau. Notre travail futur va porter sur la formalisation et l'optimisation de cet algorithme (placement et nombre de fonctions vues par pair) qui rejoint en partie le problème du *set cover*[Fei96].

Références

[AR01] P. Druschel A. Rowstron. Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Middleware 2001*, 2001.

[BG04] S. Surana R. Karp I. Stoica B. Godfrey, K. Lakshminarayanan. Load balancing in dynamic structured p2p systems. *INFOCOM'2004*, 2004.

[Fei96] U. Feige. A threshold of ln n for set cover. *STOC 1996*, 1996.

[IS01] D. Karger M. F. Kaashoek H. Balakrishnan I. Stoica, R. Morris. Chord : A scalable peer-to-peer lookup service for internet applications. *SIGCOMM 2001*, 2001.

[KPS05] M. J. Freedman K. P. Shanahan. Locality prediction for oblivious clients. *IPTPS'05*, 2005.

[LM03] A. J. Ganesh L. Massoulie, A.-M. Kermarrec. Network awareness and failure resilience in self-organising overlay networks. *SRDS'03*, 2003.

[MC02] Y. C. Hu A. Rowstron M. Castro, P. Druschel. Exploiting network proximity in peer-to-peer overlay networks. *Technical report*, (MSR-TR-2002-82), 2002.