Construction auto-organisante de tables de hachages réparties et évaluation en conditions réelles

Sveta Krasikova, Raziel Carvajal, Heverson Borba Ribeiro, Etienne Rivière, Valerio Schiavoni

Compas'2016 : Parallélisme/ Architecture / Système
Lorient, France, du 5 au 8 juillet 2016

Construction auto-organisante de tables de hachages réparties et évaluation en conditions réelles

Sveta Krasikova, Raziel Carvajal G., Heverson B. Ribeiro, Etienne Rivière et Valerio Schiavoni

Université de Neuchâtel, Suisse

Résumé
Les principes d'auto-organisation sont adaptés aux systèmes repartis à grande échelle dans des environnements dynamiques. En utilisant des actions simples, non coordonnées et fondées seulement sur une connaissance locale du système, un ensemble de pairs peut converger de façon autonome vers un état global, comme une structure de réseau logique particulière. Dans cet article, nous proposons d'étudier et d'évaluer l'utilisation de ce type de construction appliquée à des structures de tables de hachage réparties (THR) et de les étudier en conditions pratiques. Plus spécifiquement, nous considérons l'utilisation de protocoles épidémiques (*gossip*) pour l'amorçage et le maintien de plusieurs structures de THR. Nous utilisons comme base le protocole T-Chord, qui utilise le cadre de conception de systèmes auto-organisants T-Man. Nous étendons ces travaux à la construction de deux THRs supplémentaires, T-Pastry et T-Kademlia. Contrairement aux travaux précédemment publiés sur ce sujet, nous fondons notre évaluation sur un déploiement d'un prototype sur un cluster formé de 600 pairs. Nos résultats montrent que, tout en atteignant de façon systématique la structure idéale de la THR considérée, les mises en œuvre auto-organisantes tolèrent un dynamisme plus grand (va-et-vient) des pairs de la THR, et montrent que les solutions fondées sur les protocoles épidémiques ont un intérêt concret pour un déploiement en conditions réelles.

Mots-clés : table de hachage répartie, auto-organisation, protocoles épidémiques

1. Introduction

L'échelle et complexité des systèmes repartis a augmenté de manière significative au cours de la dernière décennie. Parmi les outils utilisés pour la construction de systèmes repartis à grande échelle, les tables de hachage réparties (THR, ou DHT en anglais) se sont imposées comme un bloc de base fondamental. Celles-ci permettent effectivement de mettre en œuvre de nombreuses applications allant de la diffusion en continu [1], au stockage distant [3] et aux systèmes de fichiers [2].
Les THRs offrent un service d'indexation reparti. Les pairs participants s'insèrent dans une structure de réseau logique pré-déterminée et chacun est responsable pour une partie de l'index complet. Un mécanisme de *routage* permet de parcourir la structure du réseau logique afin de rejoindre de façon déterministe le pair en charge d'un objet donné–ceci typiquement en utilisant un hachage de l'identifiant de cet objet, voire de son contenu [2]. Les exemples classiques de THRs incluent des structures fondées sur des anneaux comme Chord [14], Pastry [13] ou Tapestry [16], des structures en arbre tel que Kademlia [11] ou encore des graphes de type "papillon" pour Viceroy [9], parmi de nombreux autres.

La construction traditionnelle des THRs se fonde sur un ajout explicite des nouveaux pairs et sur des mécanismes de réparation *à la demande*. Un nouveau pair parcourt une structure déjà existante et engrange les identifiants des pairs qui devront être ses *voisins* dans la structure (ou desquels ce pair devra être le voisin). L'insertion proprement dite nécessite des précautions qui ne sont pas nécessairement mises en évidence lorsque le système est évalué par simulation. En particulier, la mise à jour des voisinages des pairs existants doit se faire de façon coordonnée et idéalement atomique, afin de ne pas obtenir une structure dégradée lors d'ajouts concurrents. De la même façon, les procédures de réparation doivent prendre en compte tous les cas où la structure n'est pas correcte et pallier ces défauts, au risque encore une fois d'obtenir une structure dégradée lors de nombreux départs concurrents.

L'utilisation de protocoles épidémiques (ou *gossip* en anglais) a été proposée comme une alternative plus simple et plus systématique pour la construction et la maintenance de réseaux logiques dans des conditions dynamiques. Les protocoles épidémiques permettent sous des conditions précises de garantir les propriétés d'auto-organisation [4]. En effet, en convergeant de façon systématique depuis n'importe quel état non-cohérent vers un état cohérent, en l'occurence une structure correspondant à la définition, ces protocoles ne nécessitent pas de prendre en compte de manière individuelle chaque potentiel cas d'erreur ou chaque ajout de pair dans la structure comme s'il s'agissait d'un événement particulier. La mise en œuvre épidémique requiert que chaque pair dispose d'une vue limitée du système, nommée sa *vue*. Chaque participant sélectionne graduellement les pairs qui correspondent le mieux à une fonction de sélection. Cette sélection est purement locale : des échanges périodiques entre pairs permettent à chacun de ceux-ci de converger graduellement vers la *meilleure* vue, émergeant ainsi de façon globale la structure. Deux cadres de conception similaires ont été proposé mettant en œuvre ces principes, Vicinity [15] et T-Man [6]. Ceux-ci sont génériques dans le sens où ils permettent d'exprimer une fonction de sélection sous la forme d'une distance devant être minimisée, et garantissent la convergence vers les voisins minimisant cette distance parmi tous les pairs présents dans le système. Le choix d'une fonction de distance appropriée permet d'émerger de façon globale une structure, ceci en n'utilisant que les actions locales que sont les échanges périodiques entre pairs.

Les auteurs de T-Man [6] ont proposé la construction de la structure de la THR Chord [14] comme exemple dans leurs travaux, sous le nom de T-Chord. Leur évaluation est néanmoins fondée sur des simulations seulement et ne permet pas de conclure sur l'adéquation des protocoles épidémiques pour la construction de cette structure en présence de fort dynamisme. Les travaux décrits dans cet article visent à étudier dans des conditions plus proches d'un déploiement réel si cette intuition de robustesse est vraie, et quel est l'impact sur la vitesse de construction et la performance de l'utilisation de ces techniques dans un prototype réellement déployé. De plus, afin d'évaluer la généralité de l'approche, nous proposons deux nouvelles constructions de THR fondées sur les principes épidémiques, T-Pastry et T-Kademlia, reproduisant de façon auto-organisante les structures des THRs Pastry [13] et Kademlia [11].

Travaux connexes. L'étude de la performance et du déploiement des THR a amené une quantité de travaux au cours de la dernière décennie. Certains de ces travaux ont évalué les performances des THRs construites de façon traditionnelle en présence de dynamisme (comme [11] pour la performance du routage dans Kademlia, ou [12] pour la résistance de la THR Bamboo en présence de dynamisme). Les travaux sus-cités sur les protocoles épidémiques [6, 15] ou des travaux similaires [5, 7] se fondent exclusivement sur des simulations et considèrent peu l'effet du dynamisme et de la concurrence en conditions réelles. LayStream [10] est un protocole de diffusion en continu fondé sur des principes épidémiques et évalué en condition réelles. Ce système n'utilise cependant pas de THR et n'est pas soumis à du dynamisme. Nous

ne connaissons pas de travaux évaluant en conditions réelles l'efficacité de la construction de THR de façon épidémique, ce qui montre l'intérêt des travaux présentés dans cet article.

2. Construction épidémique de réseaux logiques

Nous commençons par introduire brièvement les principes du cadre de conception de protocoles épidémiques de construction de réseau logique T-Man [6], utilisé pour construire les différentes THRs.
L'objectif d'une instance T-Man est de construire pour chaque participant du réseau un voisinage de pairs, généralement de taille bornée à c voisins. Ces c voisins doivent être sélectionnés selon une fonction de distance, s'appliquant à un descriptif (ou sémantique) des pairs : une fois la vue stabilisée, celle-ci doit contenir les pairs pour lesquels la distance est minimale, parmi tous les pairs présents dans le réseau. En cas d'égalité (par exemple une distance nulle) d'autres critères peuvent être employés ou simplement un choix aléatoire.
L'obtention de nouveaux candidats pour être potentiellement inclus dans la vue d'un pair se fait de manière épidémique. Périodiquement, chaque pair contacte un de ses voisins et échange sa connaissance locale du réseau (sa vue). De manière autonome, chaque partenaire de l'échange conserve les c pairs les plus proches selon la mesure de distance appliquée aux pairs présents dans les vues des deux partenaires. La garantie de convergence de T-Man repose sur l'utilisation d'un service d'échantillonnage de pairs, ou *peer sampling service* en anglais (PSS) [7]. Le PSS est lui-même mis en œuvre de façon épidémique, et construit un réseau logique sous-jacent fortement semblable à un graphe aléatoire, qui est mis à jour continuellement. Le flux de nouveaux pairs obtenus du PSS peut lui aussi être considéré pour inclusion dans la vue, par exemple lors de l'amorçage de celle-ci. La partie aléatoire que le PSS apporte est fondamentale pour garantir les propriétés de convergence. En effet, elle permet d'éviter de rester bloqué dans un minima local où un pair n'apprendrait plus de nouveaux voisins par ses interactions avec ses voisins existants, ce alors que la vue qu'il possède n'est pas optimale car des pairs plus "proches" existent ailleurs dans le système. L'utilisation du PSS permet ainsi de garantir l'obtention du minimum global [15].
Afin de garantir que les pairs ayant quitté le réseau ne soient plus visibles dans les vues de pairs restants, les entrées dans les vues sont associées à un *âge* Cet âge indique les nombre de cycles d'échanges écoulés depuis la création originale du lien. La création d'un lien vers un pair p ne peut effectivement être faite que par le pair p lui-même. Lors de la sélection du partenaire pour un des échanges réguliers, le lien avec l'âge le plus élevé est choisi. Une absence de réponse entraîne la destruction du lien, assurant la disparition des liens *morts* en un nombre fini de cycle d'échanges et, comme les pairs disparus ne génèrent plus de liens vers eux-même, leur suppression de toutes les vues en un nombre fini de cycles. À la fin de l'échange, les âges sont incrémentés pour toutes les entrées de la vue du pair qui a initié l'échange.
Exemple de construction d'un anneau. La Figure 1 illustre la construction d'une structure d'anneau, qui est à la base des THRs T-Chord et T-Pastry. Chaque nœud est identifié par un *id* choisi à partir d'un espace circulaire (par exemple, $[0..2^m - 1]$). Nous définissons la distance *cw* comme la distance sur l'anneau dans le sens des aiguilles d'une montre, c'est à dire, $cw(p.id, q.id) = (2^m + q.id - p.id) \bmod 2^m$. Comme dans cet exemple la taille de la vue est $c = 3$, l'objectif du nœud *p* est de converger vers les trois voisins qui le succèdent dans l'anneau et minimisent donc *cw*. Cette convergence résulte de l'échange avec les voisins de *p* actuels, par exemple *q* qui permet de connaître l'existence du troisième nœud de la vue finale.
Création de la structure de T-Chord. La version auto-organisante de Chord, T-Chord décrite dans [6] est fondée sur l'anneau et *cw* (création des successeurs), ainsi qu'une fonction de dis-

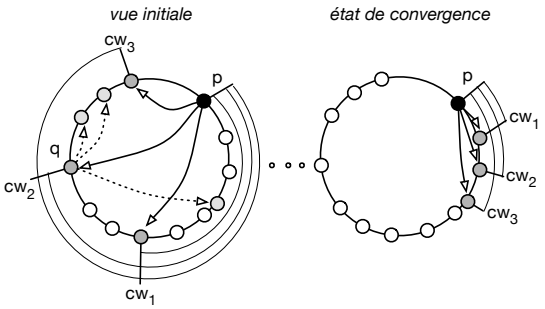


FIGURE 1 – Construction d'un anneau.

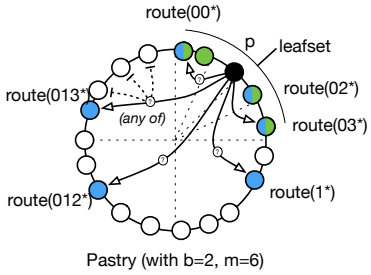


FIGURE 2 – Structure de la THR Pastry.

tance inversée *ccw* (sens trigonométrique) afin de construire le prédécesseur, et enfin une série de m instances de T-Man dont le but est pour chacune d'elle de construire un lien à longue distance (*finger*) permettant le routage efficace. La distance utilisée dans ce cas est de nouveau *cw*, mais non par rapport à *p.id* mais par rapport à une position située à mi-chemin de l'anneau pour le plus long finger (instance $m-1$), à la moitié du chemin entre *p.id* et cette destination pour le deuxième finger (instance $m-2$), et ainsi de suite. Le pair successeur de cette position sera enfin choisie comme lien pour le finger correspondant.
Nous notons dès à présent que la création de la structure permet de distinguer deux types de voisins : les voisins indispensables, qui sont nécessaire à la correction du routage–il s'agit ici des successeurs dans l'anneau, et les liens facultatifs, qui permettent d'accélérer le routage mais ne sont pas nécessaire à sa correction–ici les liens longue distance.

3. Construction de T-Pastry et T-Kademlia

Nous décrivons à présent la construction des structures de T-Pastry et T-Kademlia à l'aide d'instances de T-Man. La Table 1 résume les instances de T-Man utilisées pour chaque THR. Nous discutons ensuite brièvement des optimisations possibles dans l'interaction entre ces instances et le PSS.
T-Pastry. T-Pastry reproduit la structure de Pastry [13]. Comme pour T-Chord, celle-ci est fondée sur le principe de l'anneau précédemment présenté. Deux instances sont utilisées : une pour les liens dans le sens horaire (avec *cw*) et une pour les liens dans le sens trigonométrique (avec *ccw*). Dans les deux cas, $c = 8$ et ces deux vues forment le *leafset*. Les identifiants *p.id* des pairs dans Pastry sont vus comme des nombres en base 2^b dans l'espace circulaire de taille 2^m. Dans l'exemple de la figure 2, $b = 2$, ce qui donne des identifiants en base 4. Pour chaque préfixe de son identifiant *p.id*, le pair *p* doit connaître au moins un autre pair partageant le même préfixe, mais avec une valeur différente pour le *digit* qui le suit immédiatement. L'ensemble de ces pairs forment la table de routage qui contient au plus $\lceil \frac{m}{b} \rceil \times (2^b - 1)$ entrées. Contrairement au liens longue distance de T-Chord, où seul un pair est optimal pour chacun des liens longs, plusieurs options sont possibles pour une des entrées de la table pour Pastry (i.e., n'importe lequel des pairs avec un préfixe correspondant). Ce type de choix permet une convergence plus rapide lors de la construction auto-organisante. Bien que le nombre d'entrées dans la table de routage soit élevé, et qu'en théorie il est nécessaire d'utiliser une instance de T-Man pour chacune de ces entrées (plus 2 pour le *leafset*, soit $2 + \lceil \frac{m}{b} \rceil \times (2^b - 1)$ instances en total), en pratique il est possible de n'activer de façon automatique que celles nécessaires. Ceci est possible par exemple lorsqu'un échange indique qu'une zone n'est pas suffisamment *couverte* par des liens

Vue	Fonction de distance	Taille	Criticité
T-Pastry ($2 + \lceil \frac{m}{b} \rceil \times (2^b - 1)$ instances)			
leafset (sens horaire)	*cw* de p.id	8	indispensable
leafset (sens trigonométrique)	*ccw* de p.id	8	indispensable
$\lceil \frac{m}{b} \rceil \times (2^b - 1)$ instances : RT_π, $\pi \in$ q.id.$\mathtt{pre}(\{1..\lceil \frac{m}{b} \rceil\}) \times \{1..2^b - 1\}$)	*pd* (pour chaque préfixe π_i de p.id)	1 par entrée	facultatif
T-Kademlia (m instances)			
par *bucket* B_i avec $i \in [0..m-1]$	*xor* (pour chaque préfixe π_i de p.id, plus un pour le bit qui varie)	3 en total : 1 indispensable et 2 facultatifs	

TABLE 1 – Nombre d'instances de T-MAN pour T-Pastry et T-Kademlia.

longs (i.e., il existe des préfixes plus longs pour lesquels il n'y a pas encore de voisin connu).

T-Kademlia. La structure de la THR Kademlia [11] peut être considérée du point de vue de chaque pair p d'identifiant $p.id$ comme un arbre binaire de hauteur égale à m et pour lequel seules les branches correspondant à des préfixes de $p.id$ sont développées. Pour chacun des sous-arbres correspondant à des préfixes de $p.id$ suivis d'un bit différent de celui de $p.id$ pour le même rang, et nommé un *bucket*, un pair mandataire dont la préfixe correspond doit être trouvé. Cette technique permet de mettre en œuvre un routage fondé sur la distance du *ou exclusif*. La mise en œuvre de T-Kademlia sous forme d'instances de T-Man est assez directe : nous utilisons une fonction de distance *bd* qui vaut 0 si le préfixe correspond et la distance *ou exclusif* lorsque ce n'est pas le cas, permettant de graduellement converger vers un voisin qui possède le préfixe attendu. Une instance de T-Man est nécessaire pour chaque niveau de l'arbre binaire soit un total de m instances. Notons que pour chaque bucket nous ne maintenons pas qu'un seul voisin mais trois : le premier est considéré comme lien indispensables tandis que les deux autres sont des liens facultatifs.

Améliorations et interdépendance avec le protocole PSS. Comme précédemment indiqué, les instances T-Man utilisées pour construire les différentes THRs dépendent de la présence du PSS pour garantir la convergence. De plus, de nombreuses instances de T-Man sont présentes pour construire les différentes vues, correspondant à des fonctions de distance différentes. Nous mettons en œuvre les optimisations suivantes. Les pairs découverts au niveau du PSS sont automatiquement considérés pour inclusion dans les vues des instances T-Man sur le même pair. Les échanges pour une instance T-Man donné permettent de recevoir de nouveaux pairs candidats : dans ce cas ceux-ci sont évalués pour toutes les instances localement présentes. De nombreuses autre optimisations sont possibles, que nous laissons pour l'instant comme perspective afin d'évaluer une version simple et directement calquée sur les protocoles épidémiques publiés et évalués par simulation.

4. Évaluation

Nous évaluons une mise en œuvre (en Lua) des trois THRs à l'aide de Splay [8], un outil qui permet de simplifier le développement, le déploiement et le contrôle expérimental de systèmes répartis. Nous utilisons un cluster de 12 machines (Intel dual core, 2 Go de mémoire vive) sur un réseau switché Gigabit Ethernet, sur lequel nous déployons 600 pairs. Nous utilisons la mise en œuvre du PSS de [7] et les paramètres qui y sont recommandés (vue de taille $c = 10, S = \frac{c}{2} - 1 = 4, H = 1$). Chaque instance T-Man initie un échange tous les 5 secondes avec un partenaire choisi soit dans la vue de l'instance en question, soit depuis le PSS, avec une probabilité égale. Nous utilisons le support de Splay pour rejouer des traces de va-et-vient de

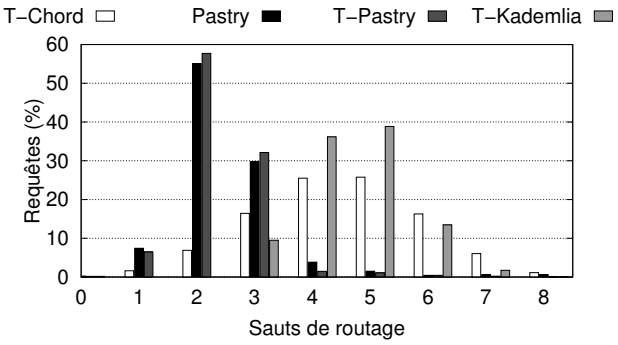
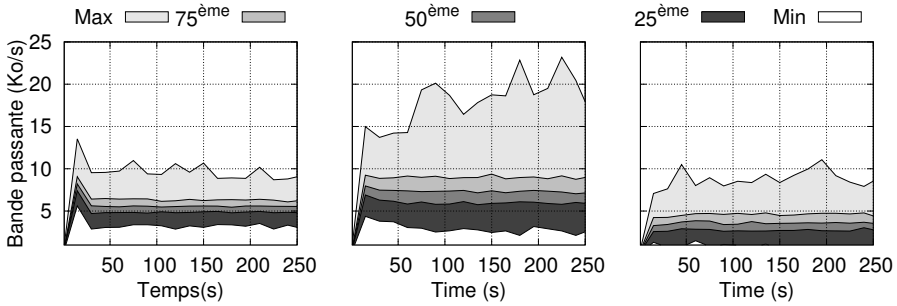


FIGURE 3 – Efficacité du routage.



(a) T-Chord (b) T-Pastry (c) T-Kademlia

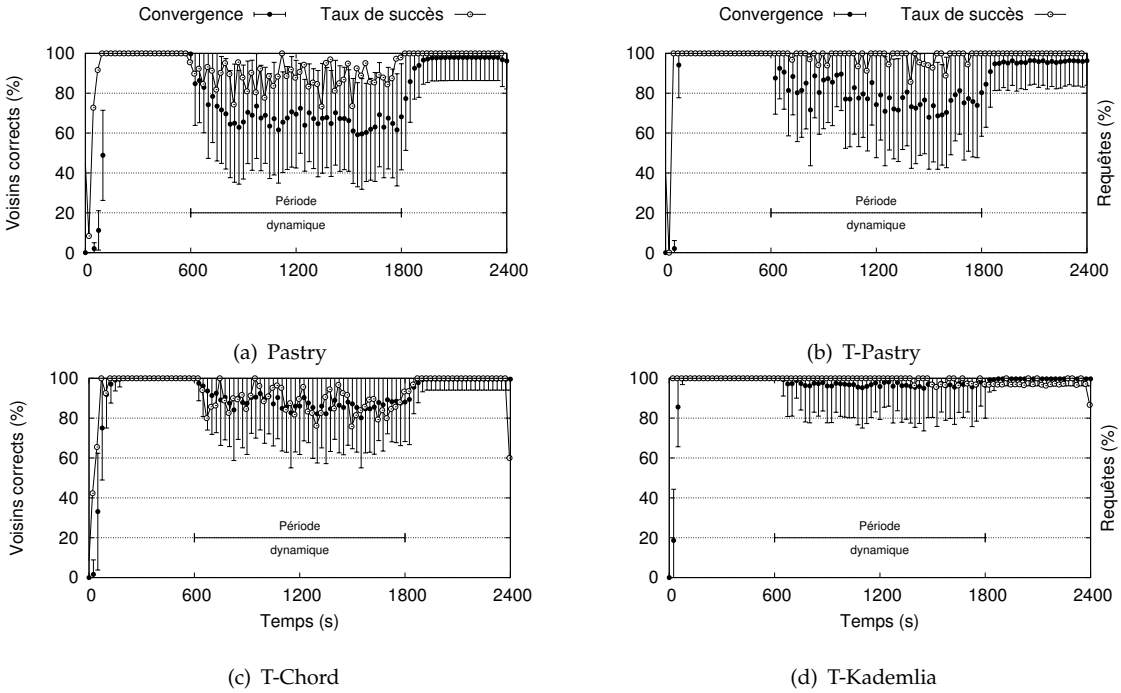
FIGURE 4 – Bande passante en envoi.

pairs dans le système et évaluer le comportement des structures des 3 THRs dans ce contexte.

Efficacité du routage. La principale métrique de performance d'une structure de THR est l'efficacité du routage dans un réseau statique ayant atteint la convergence. Nous comparons celle-ci pour les 3 THR ainsi que pour une mise en œuvre du protocole Pastry original à titre de comparaison. Nous utilisons un total de 30,000 requêtes vers des clés aléatoires qui sont effectués dès que chaque pair rejoint la THR. La figure 3 montre que la distribution du nombre de sauts suit ce qui est attendu pour un réseau de cette taille. En particulier, le nombre de sauts pour Pastry est légèrement plus faible que pour T-Pastry : ceci est du au fait que pour le premier, la structure optimale est atteinte tandis que la construction itérative du deuxième peut laisser des entrées de la table de routage vides alors qu'un pair correspondant existe néanmoins.

Consommation de bande passante. Les protocoles épidémiques sont parfois considérés comme plus coûteux que les protocoles de construction de réseau logique traditionnels. En effet, ils entraînent une consommation régulière de bande passante de part les échanges réguliers qu'ils nécessitent sur le réseau. Nous montrons néanmoins dans la figure 4 que ce besoin, si il est effectivement plus élevé que pour un réseau construit explicitement dans un contexte statique, reste raisonnable au vu des capacités réseaux actuelles. La figure 4 se lit de la façon suivante : la nuance la plus claire de gris est la valeur maximal observé pour la seconde considéré, la valeur en bas est le minimum. Les teintes de gris intermédiaires représentes les 25e, 50e (médiane) et 75e *percentiles*. Nous notons par ailleurs, notre période d'échange de 5 seconds est particulièrement agressive et peut aisément être augmenté dans un système peu dynamique pour réduire cette consommation.

Scénario dynamique. Nous évaluons le comportement des trois THRs auto-organisantes ainsi



(a) Pastry (b) T-Pastry

(c) T-Chord (d) T-Kademlia

FIGURE 5 – Convergence des voisins indispensables dans un contexte dynamique.

que de Pastry en présence de va-et-vient de pairs (système dynamique). Le système est stable pendant 20 minutes, puis pendant 20 minutes 15% des pairs sont remplacés *chaque minute* (un remplacement équivaut à un pair supprimé et un nouveau pair ajouté). Ceci est suivi de nouveau d'une période stable de 20 minutes. La figure 5 montre le taux de liens corrects (pointant vers des nœuds qui sont présents dans le système) ainsi que le taux de succès des requêtes de routage au cours du temps. Il apparaît que T-Kademlia est la structure la moins affectée par le dynamisme, tandis que T-Pastry ne présente pas une amélioration significative par rapport à sa version non auto-organisante pour ce qui est du nombre de liens corrects (néanmoins, le routage reste plus efficace pour ce dernier). La bonne performance de T-Kademlia s'explique par l'utilisation de plusieurs liens par *bucket* ou sous arbre de préfixe différent, et par le fait que la convergence (évaluée séparément et non montrée pour des raisons d'espace) est plus rapide avec ce système. La contre performance de T-Pastry s'explique par le fait qu'un grand nombre d'instances T-Man sont nécessaires et que la convergence est relativement lente en comparaison ; ainsi que par le fait que T-Pastry doive maintenir un nombre important de liens obligatoires de voisinage immédiat, augmentant la probabilité qu'une partie de ceux-ci soit défaillants.

5. Conclusion

Nous avons présenté une évaluation de l'utilisation de protocoles épidémiques de construction de réseaux logiques appliquée à 3 structures de tables de hachage réparties, 1 précédemment publiée et évaluée par simulation et deux qui sont notre contribution. Notre évaluation en conditions réelles et dynamiques montre que ce type d'approche peut être viable en pratique, bien que nous ayons aussi identifié des points sur lesquels des études plus poussés et probablement des améliorations algorithmiques allant au delà de l'utilisation directe des cadres de conception existants seront nécessaires.

Remerciements

Les travaux présentés dans cet article ont été financés par CHIST-ERA dans le contexte du projet DIONASYS et par le fond 155249 du Fond National Suisse (FNS).

Bibliographie

1. Castro (M.), Druschel (P.), Kermarrec (A.-M.), Nandi (A.), Rowstron (A.) et Singh (A.). – Splitstream : high-bandwidth multicast in cooperative environments. – In *SOSP '03 : Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 298–313, New York, NY, USA, 2003. ACM Press.
2. Dabek (F.), Kaashoek (M. F.), Karger (D.), Morris (R.) et Stoica (I.). – Wide-area cooperative storage with cfs. – In *Proceedings of SOSP '01*, pp. 202–215, 2001.
3. DeCandia (G.), Hastorun (D.), Jampani (M.), Kakulapati (G.), Lakshman (A.), Pilchin (A.), Sivasubramanian (S.), Vosshall (P.) et Vogels (W.). – Dynamo : amazon's highly available key-value store. – In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, *SOSP '07*, SOSP '07, pp. 205–220, New York, NY, USA, 2007. ACM.
4. Dolev (S.). – *Self-stabilization*. – MIT Press, 2000.
5. Jelasity (M.), Montresor (A.) et Babaoglu (O.). – Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, vol. 23, n3, 2005, pp. 219–252.
6. Jelasity (M.), Montresor (A.) et Babaoglu (O.). – T-man : Gossip-based fast overlay topology construction. *Computer networks*, vol. 53, n13, 2009, pp. 2321–2339.
7. Jelasity (M.), Voulgaris (S.), Guerraoui (R.), Kermarrec (A.-M.) et Van Steen (M.). – Gossip-based peer sampling. *ACM Transactions on Computer Systems (TOCS)*, vol. 25, n3, 2007, p. 8.
8. Leonini (L.), Rivière (E.) et Felber (P.). – SPLAY : Distributed systems evaluation made simple (or how to turn ideas into live systems in a breeze). – In *NSDI'09 : Proceedings of the 6th Symposium on Networked Systems Design and Implementation*, pp. 185–198. USENIX, apr 2009.
9. Malkhi (D.), Naor (M.) et Ratajczak (D.). – Viceroy : A scalable and dynamic emulation of the butterfly. – In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pp. 183–192. ACM, 2002.
10. Matos (M.), Schiavoni (V.), Rivière (E.), Felber (P.) et Oliveira (R.). – LayStream : composing standard gossip protocols for live video streaming. – In *14th IEEE International Conference on Peer-to-Peer Computing*, *IEEE P2P*, IEEE P2P, septembre 2014.
11. Maymounkov (P.) et Mazières (D.). – Kademlia : A peer-to-peer information system based on the xor metric. – In *Proceedings of IPTPS '02*, pp. 53–65, 2002.
12. Rhea (S.), Geels (D.), Roscoe (T.) et Kubiatowicz (J.). – Handling churn in a DHT. – In *Proceedings of the USENIX Annual Technical Conference*, pp. 127–140. Boston, MA, USA, 2004.
13. Rowstron (A.) et Druschel (P.). – Pastry : scalable, decentralized object location and routing for large-scale peer-to-peer systems. – In *Proceedings of Middleware'01*, pp. 329–350, novembre 2001.
14. Stoica (I.), Morris (R.), Liben-Nowell (D.), Karger (D. R.), Kaashoek (M. F.), Dabek (F.) et Balakrishnan (H.). – Chord : A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, vol. 11, n1, feb 2003, pp. 17–32.
15. Voulgaris (S.) et van Steen (M.). – Vicinity : A pinch of randomness brings out the structure. *In : Middleware 2013*, pp. 21–40. – Springer, 2013.
16. Zhao (B. Y.), Huang (L.), Stribling (J.), Rhea (S. C.), Joseph (A. D.) et Kubiatowicz (J. D.). – Tapestry : A resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, vol. 22, n1, 2004, pp. 41–53.