Lot 2
Applications

Modélisation probabiliste de Pastry

Description :	les différents acteurs des services sur réseau IP s'intéressent aux architectures de type *perr-to-peer*. Différents algorithmes de routage existent pour celles-ci, notamment l'algorithme DHT (*Distributed Hash Table*) proposé par Microsoft. Nous étudions ici ce dernier au moyen des outils PRISM et APMC.
Auteur(s) :	Guillaume Châtelet, Benoît Parreaux Richard Lasssaigne, Jean-François Monin
Référence :	Averroes / Lot 2 / Fourniture 2 / V1.0
Date :	6 août 2006
Statut :	validé
Version :	1.0

Réseau National des Technologies Logicielles
Projet subventionné par le Ministère de la Recherche et des Nouvelles Technologies

CRIL Technology, France Télécom R&D, INRIA-Futurs, LaBRI (Univ. de Bordeaux – CNRS), LIX (École Polytechnique, CNRS) LORIA, LRI (Univ. de Paris Sud – CNRS), LSV (ENS de Cachan – CNRS)

Historique

26 juillet 2006	V 0.1	version préliminaire
6 août 2006	V 1.0	mise au format averroes

Table des matières

Résumé

Dans le cadre du projet AVERROES, nous avons utilisé les outils issus des travaux de recherche des laboratoires universitaires sur des problématiques industrielles. Les différents acteurs des services sur réseau IP s'intéressent aux architectures de type P2P. Dans ce cadre, différents algorithmes de routage existent et nous souhaitions analyser plus particulièrement l'algorithme DHT (Distributed Hash Table). Notre objectif était double : confirmer une étude d'une implémentation de cette algorithme nommé Pastry [5, 14]faite par les laboratoires Microsoft [1, 13] et étudier sa stabilité selon la fréquence de l'arrivé et du départ de nœuds.

1 Présentation de l'algorithme DHT/Pastry

Le routage réalisé par l'algorithme DHT est basé sur une organisation en anneau des nœuds participant. Chaque nœud sera donc placé sur cet anneau selon la valeur d'un identifiant attribué en appliquant une fonction de hashage sur différents paramètres caractérisant ce nœud (en particulier son adresse IP). L'identifiant calculé par la fonction de hashage est définit dans une base B et peut être d'une longueur maximal M. Ce procédé de calcul des identifiant revient à découper récursivement l'anneau en B parties et de placer le nœud dans la partie correspondant à son identifiant.

Chaque nœud dispose d'une table de routage lui permettant d'accéder directement au nœud correspondant aux identifiants contenus dans cette table. En plus de cette table de routage, le nœud connaît un certain nombre de nœuds proches de lui au sens IP et un ensemble des nœuds les plus proches de lui au sens de leurs identifiants. Le routage se fait de la façon suivante : le nœud regarde si l'identifiant vers lequel il veut router n'est pas dans la liste de ses voisins au sens des identifiants (leaf). Si ce n'est pas le cas on cherche dans la table de routage un nœud ayant un préfixe commun le plus long possible avec l'identifiant vers lequel on veut router. Dans de rare cas, il est possible que l'on ne puisse pas router et dans ce cas on choisi un nœud dans l'ensemble des nœuds connus (pour des raisons de finitude le préfixe commun entre l'identifiant du nœud choisi et l'identifiant recherché doit toujours décroitre). Lorsqu'un nœud a été choisi la demande de routage est envoyé a ce dernier. La procédure est alors répétée jusqu'à trouver le nœud ayant l'identifiant recherché.

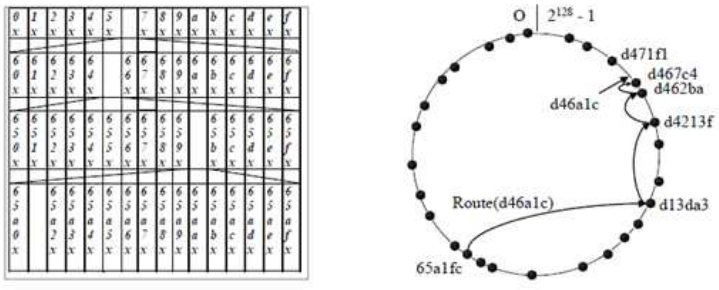

FIG. 1 – Exemple de routage et de table de routage

La Figure 1 donne un exemple du résultat du routage pour l'identifiant d46a1c a partir du nœud 65a1fc. L'insertion d'un nœud dans l'anneau est réalisée en utilisant l'algorithme de routage a partir d'un nœud proche (au sens des adresses IP) du nœud à introduire. Tous les nœuds visités lors de l'insertion voient leurs tables de routage mises à jour.

2 Modélisation

La modélisation a été réalisée en utilisant les outils développés dans le cadre du projet Averroès [2, 6]. Le but de cette modélisation est de calculer le nombre de sauts (nb_hop) nécessaires pour router. Le premier travail a été de définir le niveau d'abstraction auquel on se place pour décrire le routage. Il a été choisi d'abstraire les identifiants et de définir le routage en termes de distance entre identifiants (différence entre les identifiants). On peut alors définir l'évolution de cette distance en fonction du cas de routage dans lequel on se trouve. Lorsque le nœud cherché se trouve dans la liste des leaf, le routage est fini, il faut juste incrémenter nb_hop. Dans le cas ou l'on utilise la table de routage, la distance est divisée par B (la base) puisque la taille du préfixe diminue et nb_hop est incrémenté. Dans de rares cas la distance ne varie pas, mais nb_hop est incrémenté.

Pour modéliser l'introduction d'un nœud, on a choisi de tirer son identifiant au hasard en se plaçant dans le cas d'une distribution homogène des nœuds sur l'anneau. La distance moyenne entre deux nœuds est donc $\frac{B^l}{nb\_node}$, de ce fait $a \in \mathsf{leaf}_n \cong \frac{B^l}{nb\_node} \cdot \frac{L}{2}$ où l est le nombre de chiffres de l'identifiant et L le nombre d'éléments maximum dans l'ensemble leaf.

La modélisation de la table de routage est réalisée en utilisant le taux de remplissage des lignes de l'ensemble des tables pour calculer la probabilité qu'une entrée correspondant à l'identifiant cherché existe. Cette probabilité s'écrit de la façon suivante : $\mathtt{prob}(RT[i,j]_n) \neq \perp \approx \frac{T(i)}{B.nb\_node}$.

La mise à jour de la table se fait de la façon suivante principalement lors de l'insertion d'un nœud. Pour chaque entrée de la ligne i dans la table de n (b entrées), si x est le nombre de nœuds dans la table et dans les leaf qui partagent la même ligne d'entrée alors on tire x fois $\frac{T(i)}{B.nb\_node}$ pour savoir si l'entrée de n est remplie. Si elle est remplie on ajoute à $T(i)$ le nombre de fois que le tirage a raté plus un. De cette façon, il est possible de faire évoluer le taux de remplissage de chaque ligne dans le temps.

La modélisation probabiliste des tables de routage peut paraître un peu compliquée voir même artificielle. Cependant, il faut noter que les algorithmes distribué peer to peer ont très souvent un comportement indéterministe. En effet, les problèmes classiques liés à la distribution sont généralement résolus en introduisant des composantes aléatoires qui garantissent une bonne stabilité de ce type d'algorithme dans le temps. De ce fait, l'introduction du paradigme probabiliste est adapté à la spécification de ces algorithmes. Par ailleurs, les outils de génie logiciel classiques sont impuissant devant la compléxité relative de ces algorithmes pour nos besoins. En effet, une forte abstraction du modèle peut être manipulé par les outils classiques, mais ne permet pas d'obtenir les résultats attendus et une modèlisation assez complète pour obtenir les résultats attendu ne peut être traité par les outils classiques. La modélisation probabiliste permet d'obtenir un niveau de description sur lequel nous pouvons travailler [12, 7].

2.1 Modèle théorique

Le modèle théorique est un modèle très simple. On modélise le routage en faisant abstraction des identifiants des nœuds. Pour cela on utilise la distance entre la source et le destinataire. À chaque saut, si le système suit une loi en $\log_B$, la distance devra être divisée par la B, jusqu'à ce que cette distance soit inférieure à $\frac{B^l}{nb\_node} \cdot \frac{L}{2}$, soit que le nœud se trouve dans les leaf.

Le code de cette modélisation tient en quelques dizaines de ligne et ne contient pas particulièrement de probalité. Mais, il nous a permis d'évaluer la puissance de calcul nécéssaire à la réalisation des courbes que nous présentons dans la section suivante ainsi que de témoin en comparaison des courbes que nous avons obtenues avec le modèle suivant.

2.2 Modèlisation des tables de routage

La modélisation de toutes les tables de routages est imposible. Cependant le remplissage des tables de routage joue un rôle prépondérant dans l'algorithme du DHT. Il nous semblait important de trouver un moyen de le modélisé.

Dans ce but, nous avons remplacé les tables de routage par une probabilité de trouver l'adresse permentant le routage en $\log_B$. Cette probabilité est calculé comme suit : On calcule une estimation $T(i)$ du nombre de cases remplies sur l'ensemble des tables de routages présentes sur le réseau sur leur ligne i. Comme il y a B case par ligne et par table de routage, s'il y a nb_node nœuds sur le réseau, il y a $B.nb\_node$ case à remplir. Donc la probabilité de trouver une case remplie sur la ligne i d'une table de routage prise au hasard est $\frac{T(i)}{B.nb\_node}$.

Nous modélisont le routage des messages en utilisant cette propbabilté. En cas de réussite du tirage, une nouvelle distance est tirée au hasard en faisant en sorte que le $\log_B$ décroisse d'au moins 1. Dans le cas contraire, la distance ne bouge pas.

La partie difficile de la modélisation est la mise à jour de la table de routage, pour nous la mise à jour des valeurs $T(i)$. Évidement, cette partie est aussi la plus difficile à implémenter dans le service réel. Les tables de routages se mettent à jour lors des introductions de nœud dans le réseau : Une mise à jour pour chaque introduction de nœud. Lors de ces introduction de nœud. Il y a deux moyen possible d'ajouter une entrée dans une table de routage.

La première est une entrée venant des leaf, c'est à dire la liste des voisins immédiats d'un nœud. Cette mise à jour dépend de la mise à jour des leaf. Lors de l'introduction d'un nœud dans le réseau, les L nœuds voisins du nouveau nœud sont mis à jour. Il recoivent l'identifiant du nouveau nœud, le nouveau nœud, quant à lui, reçoit L identifiant. Nous modélisons donc $2.L$ mise à jour de la table de routage par les leaf. Cette mise à jour s'opère de la manière suivante : On tire la probabilité qu'il y ait un creux dans la table de routage, si le tirage réussit, alors le creux est comblé.

La deuxième possiblité est la mise à jour de la table de routage du nouveau nœud. Lors de l'introduction d'un nouveau nœud, l'ensemble des nœuds ayant participé à l'introduction du nœud offre au nouveau nœud leur table de routage. Celui-ci se construit sa propre table en fonction des tables qu'il recoit. Mais l'ensemble des tables qu'il reçoit ne sert pas à la mise à jour de sa table. Plus un nœud est éloigné sur l'anneau, moins leur table de routage est pertinant pour le nouveau nœud. Ainsi, il nous faut lors de la simulation du routage capturer le nombre de nœud qui sera utile au nouveau nœud pour sa mise à jour. Cette capture se fait ligne par ligne et en fonction de la distance.

Une fois cette capture faite, nous obtenons pour chaque ligne du nouveau nœud, le nombre de nœud qui participera à la mise à jour : $U(i)$. Ainsi, pour chaque entrée dans la linge i du nouveau nœud, nous avons $U(i)$ valeur venant des autres nœuds. La mise à jour est modélisée comme suit : Pour chaque entrée, on tire $U(i)$ la probabilité d'avoir une entrée en gardant en mémoire les échecs. S'il n'y a que des échecs, l'entrée du nouveau nœud reste vide (aucune mise à jour n'est possible). S'il y a une réussite, la nouvelle entrée est mise à jour et on incrémente $T(i)$ de 1. Mais, comme le nouveau nœud envoie sa table de routage à tous les nœuds ayant participé à son introduction dans le réseau, les nœud n'ayant pas d'entrée seront mise à jour. Dans notre modèle, ceci se traduit en augmentant $T(i)$ du nombre d'échec de tirage. Cette opération se répète B fois (nombre de entrée dans une ligne de table de routage) pour tous les i pour lequel $U(i)$ est diférent de 0.

3 Résultats obtenus

L'étude a été faite avec deux outils différents (PRISM [3, 11, 10] et APMC [4, 9, 8]). Le premier réalise des calculs exacts sur la spécification alors que le deuxième outil utilise une approche basée sur la simulation. Nous avons étudié deux valeurs nous paraissant intéressantes : la qualité du routage et le taux de remplissage de la table de routage.

3.1 Qualité de routage

La qualité du routage est vérifiée utilisant les courbes 2 et 3. La première montre la courbe théorique corespondant à un routage en $\log_n$. Elle représente en fonction du nombre de nœud, la répartition du nombre de *hop* lors d'un routage. Par exemple, pour un réseau contenant 100

nœuds, la probabilité pour que le routage ce passe en 3 sauts est d'environ 50%. Pour 4, saut la probalité est d'environ 35%.

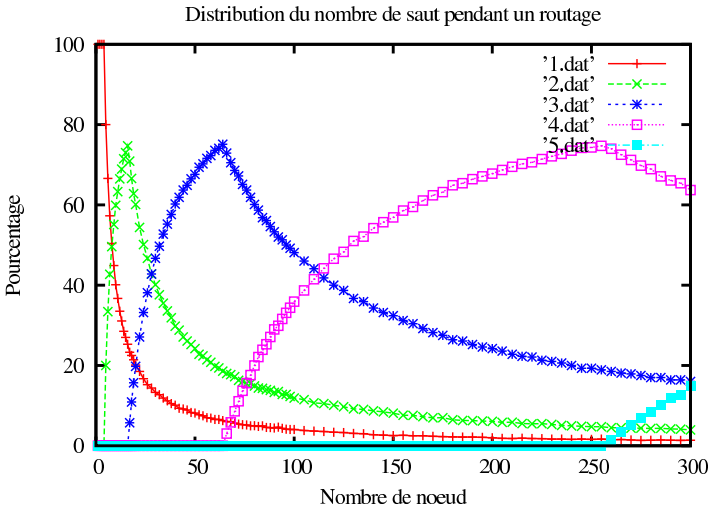

FIG. 2 – Courbe théorique

Cette courbe théorique est a mettre en parallèle avec la courbe observée à l'aide de notre modélisation du routage. La figure 3 représente les données que nous avons obtenu après simulation. On observe que les courbes s'infléchissent aussi au passage des différante valeur de $(log)_B$, c'est a dire 4,16,64 et 256. On montre donc ainsi que le nombre de sauts lors des routages suit bien une lois en $o(log_B)$.

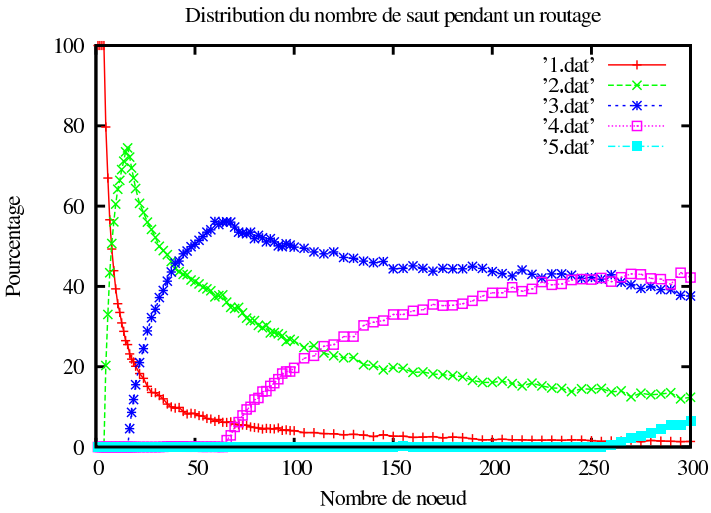

FIG. 3 – Courbe observée sur le modèle

On note une différance entre les deux courbes, la décroissance est plus faible sur les courbes observées que sur les courbes théoriques.On pourrait en déduire que le routage est meilleur que la

courbe théorique espéré. Par exemple, pour 100 nœuds, le taux de routage en 4 hop n'est que de 35% au lieu de 45% sur la courbe théorique. Ceci peut s'expliquer par le fait que notre modèle utilise une base très faible (ie 4), et il y a de grande chance (ie 1/4) que le routage saute une étape réduisant aussi le nombre de saut dans le routage. En augmenant la base (jusqu'à 16 par exemple), la différence entre les deux courbes devrait être moins impressionnante. Malheureusement, augmenter la base n'est pas une chose aisé, la limitation vient du codage des entiers dans le model-checker mais aussi l'augmentation de la taille du modèle et du temps de calcul que cela implique.

3.2 Qualité des tables de routage

La courbe 4 que nous proposons représente le taux de remplissage des tables de routage en fonction du nombre de nœud présents sur le réseau et pour chaque ligne des table.

La ligne 6 correspond à la ligne R_6 des tables de routage, c'est à dire la ligne où sont référencé des nœuds partageant 2 digits avec le posseceur de la table de routage.

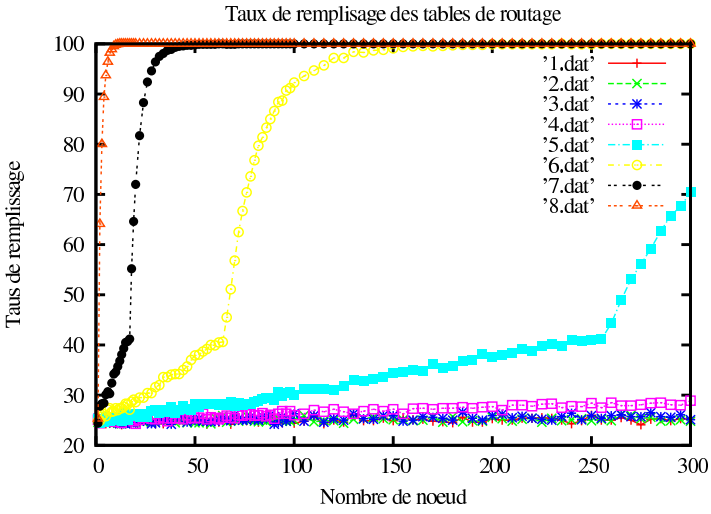

FIG. 4 – Exemple de routage et de table de routage

On observe trois phases dans le remplissage des tables de routages.

1. Mise à jour de la table de routage à partir du moment où il commence à y avoir suffisamment de nœuds sur le réseau. Pour la courbe 6, c'est à partir de 16 nœuds que la courbe commence à s'infléchir vers le haut.
2. Les voisins (leaf) viennent remplir la table lorsque le nombre d'éléments rend le remplissage de cette ligne inévitable. Pour la courbe 6, ceci intervient à partir de 64 nœud.
3. Des mise à jours ponctuelles réalisée lors de l'introduction d'un nouveau nœud permettent de finir le remplissage de la table de routage dans la suite de l'exécution.

L'interprétation de cette courbe nous permet de dire que les deux modes de mise à jour sont complémentaires et nécéssaire pour le bon fonctionnement de l'algorithme de DHT. La mise à jour de la table de routage en utilisant les voisins (leaf) permet d'accroitre le nombre d'entrée remplie dans les tables de routage très rapidement. Cependant l'accroissement du taux de remplissage s'infléchit aux environ de 80%. Intervient alors le deuxxième système de mise à jour qui va comblé les trous laisser par la « méthode leaf » afin d'atteindre les 100% de taux de remplissage.

Cette observation peu aussi être faite sur le modèle que nous avons choisit, puisque on observe qu'avec la « méthode leaf » on incrémente le nombre d'entrée lorsque le tirage de présence échoue.

Donc cette mise à jour à un role très important quand le taux de remplissage est faible. Avec la « méthode Routage », on met à jour les nœud qui ont des creux si les autres sont remplis, cette méthode à donc un grand rôle pour combler les trous. Ces observations ont été vérifiées en modélisant le système sans l'une et l'autre des méthodes de mises à jour. Les courbes ne sont pas présentées dans ce document

Ce travail de modélisation nous a donc permis d'étudier le comportement de l'algorithme en utilisant les outils au travers des résultats obtenus. Nous avons aussi tiré de cette expérience une meilleure connaissance de de ce type d'algorithme en affinant notre compréhension des mécanismes mis en place lors de la phase de spécification. En particulier, nous avons remarqué que les algorithmes distribués colaboratifs sont souvent partiellement immunisés contre les défauts d'implémentation par les mécanismes mis en œuvre en particulier ceux permettant d'assurer une bonne tolérance aux fautes. Par exemple, le DHT fournit un routage même si la « méthode leaf » n'est pas active ou que partiellement active. Les performances de ce routage sont dégradés par rapport aux posiblités du système en fonctionnement nominale mais il est quand même assuré. De ce fait, les défauts éventuel de notre modélisation peuvent être masqués bien que le modèle réagisse en accord avec les attentes théoriques du système.

De même que pour l'implémentation du modèle, l'implémentation réelle d'un algorithme de ce type peut être erronée alors que le protocole semble répondre aux attentes théoriques. Ainsi, il parrait difficile de valider l'implémentation d'un tel algorithme en utilisant des techniques de test. L'observation du comportement ne suffit pas car un comportement déviant peut être masqué par la tolérance aux fautes du système.

4 Conclusion et perspectives

Cette expérimentation nous permet de tirer plusieurs conclusions. Premièrement, même si le premier outil permet de réaliser des calculs exacts, pour l'utilisation que nous voulions en faire il ne semble pas très adapté. En effet, dès que le nombre de nœud augmente les temps de calcul deviennent exorbitants et très vite il devient impossible d'obtenir des résultats. Cependant si l'on est attaché au fait d'obtenir un résultat très précis l'utilisation de cet outil est vivement conseillé. Il faudra tout de même faire attention de ne pas devoir travailler avec un nombre de nœuds trop important.

Le deuxième outil réalise ses calculs grâce à des simulations successives jusqu'à ce que les probabilités déduites de ces simulations varient moins qu'un seuil prédéfini. Ce type d'outil présente pour notre problématique plusieurs avantages. Tout d'abord, il est possible de traiter un nombre important de nœuds. Ensuite, il est possible d'obtenir des résultats durant le calcul et donc même si l'on ne fini pas le calcul des résultats partiels sont disponibles. Nous avons obtenus des résultats très intéressant avec cet outil qui nous ont permis de vérifier les résultats obtenus par Microsoft et de montrer la stabilité du routage en utilisant DHT. Les simulations nous ont aussi permis de prendre conscience de l'importance de l'ensemble leaf dans cet algorithme.

Les outils que nous avons expérimenté donnent des résultats intéressant à la condition de maitriser les techniques de spécification et d'abstraction. Ils sont donc réservés à des spécialistes qui auraient pour but de réaliser les spécifications et l'utilisation des outils en collaboration avec les concepteurs des systèmes. Pour nos besoins, l'outil APMC semble mieux adapté mais cela ne préjuge en rien de l'intérêt de l'autre outil.

De plus la phase de modélisation nous a permis de mieux comprendre les mécanismes mis en jeux dans la mise en place des tables de routage de l'algorithme du DHT. En particulier, les rôles complémentaire des deux méthodes de mise à jour. et de prendre conscience de la porté des mécanismes mis en place pour assurer la stabilité de l'algorithme sur son comportement.

Nous pensons qu'il pourrait être très intéressant de voir comment il serait possible d'utiliser ces outils en compléments de résultats d'expérimentation ou avec les résultats obtenus grâce à un simulateur. Ce système pourrait être particulièremment adapté à la validation d'algorithmes distribués, en particulier, ceux qui présentent des propriétés de tolérance aux fautes, auto-stablisation, etc. Ceci d'autant que, comme nous l'avons souligné, ce type de protocoles peut être difficile à

valider par des techniques de test.

Références

[1] `http://research.microsoft.com/~antr/Pastry.htm`.

[2] `http://www-verimag.imag.fr/AVERROES/`.

[3] `http://www.cs.bham.ac.uk/~dxp/prism/`.

[4] `http://apmc.berbiqui.org/index.php/Accueil`.

[5] M. Castro, P. Druschel, Y.C. Hu, and A Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. Tech. Rep. MSR-TR-2002-82,, Microsoft Research, One Microsoft Way, Redmond, WA 98052, 2002.

[6] M. Duflot, L. Fribourg, Th. Hérault, R. Lassaigne, F. Magniette, S. Messika, S. Peyronnet, and C. Picaronny. Verification of the CSMA/CD protocol using PRISM and APMC. In *Proc. 4th Int. Workshop on Automated Verification of Critical Systems (AVoCS 2004)*, London, UK, September 2004. Electronic Notes in Theor. Comp. Sci.

[7] C. Courcoubetis et M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 24(4) :857–907, 1995.

[8] G. Guirado, T. Hérault, R. Lassaigne, and S. Peyronnet. Distribution, approximation and probabilistic model checking. In *Proc. of the 4th Parallel and Distributed Methods in Verification (PDMC 05)*, Lisboa, Portugal, 2005. Electronic Notes in Theor. Comp. Sci.

[9] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate Probabilistic Model Checking. In *5th Verification, Model Checking and Abstract Interpretation (VMCAI 2004)*, volume 2937 of *LNCS*, pages 73–84, Venice, Italy, 2004. Springer.

[10] Andrew Hinton, Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM : A Tool for Automatic Verification of Probabilistic Systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, March 2006.

[11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 2.0 : A Tool for Probabilistic Model Checking. In *Proc. 1st International Conference on Quantitative Evaluation of Systems (QEST'04)*, pages 322–323. IEEE Computer Society Press, September 2004.

[12] R. Lassaigne and S. Peyronnet. Approximate verification of probabilistic systems. In *2nd joint Process Algebra and Performance Modelling and Probabilistic Methods in Verification (PAPM-PROBMIV 2002)*, volume 2399 of *LNCS*, pages 213–214. Springer, 2002.

[13] A. Rowstron and P. Druschel. Pastry : Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware'2001*, Germany, November 2001.

[14] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. In *ACM SOSP*, October 2001.