Raisonnement distribué dans un environnement de type Pair-à-Pair

P. Adjiman P. Chatalic F. Goasdoué M.-C. Rousset L. Simon *
{adjiman,chatalic,fg,mcr,simon}@lri.fr

Résumé

Dans un système d'inférence pair-à-pair, chaque pair peut raisonner localement mais peut également solliciter son voisinage constitué des pairs avec lesquels il partage une partie de son vocabulaire. Dans cet article, on s'intéresse aux systèmes d'inférence pair-à-pair dans lesquels la théorie de chaque pair est un ensemble de clauses propositionnelles construites à partir d'un vocabulaire local. Une caractéristique importante des systèmes pair-à-pair est que la théorie globale (l'union des théories de tous les pairs) n'est pas connue (par opposition aux systèmes de raisonnement fondés sur le partitionnement). La contribution de cet article est double. Nous exposons le premier algorithme de calcul d'impliqués dans un environnement pair-à-pair: il est anytime et calcule les impliqués progressivement, depuis le pair interrogé jusqu'aux pairs de plus en plus distants. Nous énonçons une condition suffisante sur le graphe de voisinage du système d'inférence pair-à-pair, garantissant la complétude de notre algorithme. Nous présentons également quelques résultats expérimentaux prometteurs.

1 Introduction

Depuis quelque temps, les systèmes pair-à-pair font l'objet d'une attention considérable, en raison de leur architecture sous-jacente, particulièrement adaptée au passage à l'échelle des applications distribuée que l'on trouve sur Internet. Dans un système pair-à-pair, il n'y a aucun contrôle centralisé ni aucune organisation hiérarchique: chaque pair est équivalent en fonctionnalité et coopère avec d'autres pairs dans l'objectif de réaliser une tâche collective. Les systèmes pair-à-pair ont évolué depuis le simple système de partage de fichiers distribués (avec interrogation par mots clés) comme Napster [Nap] et Gnutella [Gnu] jusqu'aux systèmes évolués de gestion de données distribuées comme Edutella [NWQ+] ou Piazza [HITM], qui offrent un niveau sémantique de description des données permettant l'expression de requêtes complexes. Dans ces systèmes, la complexité du problème de la réponse à une requête est directement lié à l'expressivité du formalisme utilisé pour mettre en relation les schémas des différents pairs via des correspondances sémantique (mappings) [HIST].

*Pôle Commun de Recherche en Informatique du plateau de Saclay, CNRS, École Polytechnique – X, INRIA et Université Paris Sud.

Dans cet article, nous nous intéressons aux systèmes d'inférence pair-à-pair, dans lesquels chaque pair peut répondre à une requête en raisonnant sur sa théorie (propositionnelle) locale mais également en posant des requêtes à d'autres pairs avec lesquels il est sémantiquement relié, via un partage d'une partie de son vocabulaire.

Cet environnement englobe de nombreuses applications telles que les systèmes d'intégration d'information pair-à-pair ou les agents intelligents dans lesquels chaque pair possède sa propre connaissance (sur ses données ou son domaine d'expertise) et une connaissance partielle des autres pairs. Dans ces conditions, lorsqu'un pair est sollicité pour exécuter une tâche de raisonnement, si il ne peut pas la résoudre localement dans sa totalité, il doit être capable de la découper en sous tâches à distribuer aux pairs appropriés de son voisinage. Ainsi, étape par étape, la tâche initiale est découpée puis distribuée aux pairs compétents pour en résoudre une partie. Les réponses de chaque sous tâches doivent être ensuite recomposées pour construire la réponse de la tâche initiale.

Nous nous intéressons aux systèmes d'inférence pair-à-pair dans lesquels la théorie de chaque pair est un ensemble de clauses propositionnelles construites à partir d'un ensembles de variables propositionnelles représentant son vocabulaire local. Chaque pair peut partager une partie de son vocabulaire avec d'autres pairs. Nous considérons la tâche de raisonnement qui consiste à trouver les conséquences d'une certaine forme (par exemple des clauses ne contenant seulement que certaines variables) pour une formule donnée en entrée exprimée en fonction du vocabulaire local du pair interrogé. Notons que d'autres tâches de raisonnement comme la recherche d'impliquants d'une certaine forme pour une formule donnée peut être réduite par équivalence à une tâche de recherche de conséquences.

La contribution de cet article est double. Nous présentons le premier algorithme de calcul d'impliqués dans un environnement pair-à-pair: il est anytime et calcule les impliqués progressivement, depuis le pair interrogé jusqu'aux pairs de plus en plus distants. Nous énonçons une condition suffisante sur le graphe de voisinage du système d'inférence pair-à-pair garantissant la complétude de notre algorithme

Soulignons que le problème de raisonnement distribué étudié dans cet article est différent du problème de raisonnement sur des théories partitionnée obtenus par décomposition ([DR, AM]). Dans ce dernier problème, une théorie centralisée (relativement grande) est donnée, et sa structure est exploitée afin de calculer la meilleure façon de la découper, dans l'objectif d'optimiser un algorithme de raisonnement travaillant sur des théories partitionnées. Dans notre problème, la théorie complète (i.e., l'union de toutes les théories locales) n'est pas connue et la partition est imposée par l'architecture pair-à-pair elle même. Comme nous allons l'illustrer sur un exemple, (section 2), les algorithmes fondés sur la transmission de clauses entre partitions (dans l'esprit de [AM, DR, dV]) ne sont pas appropriés pour notre problème de recherche de conséquences. Notre algorithme découpe les clauses si elles contiennent des variables appartenant à des vocabulaires de différents pairs. Chaque morceau d'une clause découpée est ensuite transmis à la théorie concernée pour calculer ses conséquences. Les conséquences de chaque morceau doivent être recombinées pour produire les conséquences de la clause qui a été découpée.

L'article est organisé de la manière suivante. La section 2 définit formellement le problème d'inférence pair-à-pair auquel on va s'intéresser. Dans la section 3, nous décrivons un algorithme distribué de calcul de conséquences et nous énonçons ses propriétés. La section 4 décrit quelques résultats expérimentaux. La section 5 situe ce travail par rapport à d'autres approches existantes. La section 6 conclut par une brève discussion.

2 Inférence pair-à-pair: définition du problème

Un système d'inférence pair-à-pair (P2PIS) est un réseau de théories appartenant à des pairs. Chaque pair P est un ensemble fini de formules propositionnelles du langage $\mathcal{L}_P$. On considère que $\mathcal{L}_P$ est le langage des clauses sans répétitions de littéral que l'on peut construire à partir d'un ensemble fini de variables propositionnelles $\mathcal{V}_P$, appelé le *vocabulaire* de P.

Les pairs peuvent être sémantiquement connectés en ayant des variables en commun dans leur vocabulaire respectifs, appelés *variables partagées*. Dans un P2PIS, aucun pair n'a la connaissance de la théorie globale du P2PIS. Chaque pair connaît seulement sa théorie locale et sait qu'il partage certaines variables de son vocabulaire avec d'autres pairs du P2PIS (son *voisinage*). Il ne connaît pas nécessairement *tous* les pairs avec lesquels il partage des variables. Quand un nouveau pair se connecte au P2PIS, il se déclare simplement auprès des pairs avec lesquels il veut partager des variables. Un P2PIS peut être formalisé par un *graphe de voisinage*.

Définition 1 (Graphe de voisinage) *Soit $\mathcal{P} = (P_i)_{i=1..n}$ une famille de théories sous formes clausale construites sur leurs vocabulaires respectifs $\mathcal{V}_{P_i}$, soit $\mathcal{V} = \cup_{i=1..n}\mathcal{V}_{P_i}$. Un graphe de voisinage est un graphe $\Gamma = (\mathcal{P}, \text{VOIS})$, où $\mathcal{P}$ est l'ensemble des nœuds et $\text{VOIS} \subseteq \mathcal{V} \times \mathcal{P} \times \mathcal{P}$ est un ensemble d'arètes étiquetées, tel que pour chaque $(v, P_i, P_j) \in \text{VOIS}$, $i \neq j$ et $v \in \mathcal{V}_{P_i} \cap \mathcal{V}_{P_j}$.*

Une arête étiquetée (v, P_i, P_j) exprime que les pairs P_i et P_j savent tous deux qu'ils partagent la variable v. Pour un pair P et un littéral l donnés, $\text{VOIS}(l, P)$ dénote l'ensemble des pairs partageant la variable l avec P.

Pour chaque théorie P, on considère un sous ensemble de *variables cibles* $\mathcal{VC}_P \subseteq \mathcal{V}_P$, supposé représenter les variables d'intérêt pour l'application, (e.g., les faits observables, dans une application de diagnostic fondée sur les modèles, ou les classes représentant les données, dans une application d'intégration d'informations). Pour une clause donnée (appelée la *requête*) en entrée d'un pair du P2PIS, l'objectif est de calculer toutes ses conséquences (appelées *réponses*) dont les variables appartiennent au *langage cible*.

Remarquons que même si la requête est exprimée en termes du vocabulaire local du pair interrogé, les réponses attendues peuvent elles contenir des variables cibles de différents pairs. Les langages cibles considérés par notre algorithme sont définis en termes de variables cibles et nécessitent qu'une variable partagée aie le même statut de cible dans tous les pairs qui la partagent.

Définition 2 (Langage Cible) *Soit $\Gamma = (\mathcal{P}, \text{VOIS})$ un P2PIS et pour chaque pair P, soit $\mathcal{VC}_P$ l'ensemble de ses variables cibles tel que si $(v, P_i, P_j) \in \text{VOIS}$ alors $v \in \mathcal{VC}_{P_i}$ ssi $v \in \mathcal{VC}_{P_j}$. Pour un sous ensemble SP de pairs de $\mathcal{P}$, on définit son langage cible $\mathcal{C}ible(SP)$ comme le langage des clauses (incluant la clause vide) construites avec uniquement des variables de $\bigcup_{P \in SP} \mathcal{VC}_P$.*

Parmi les réponses possibles, on distingue les *réponses locales*, faisant intervenir uniquement les variables locales du pair interrogé, les *réponses distantes*, faisant intervenir les variables d'un seul pair, et les *réponses d'intégration* faisant intervenir des variables cibles de plusieurs pairs.

Définition 3 (Impliqués premiers propres relativement à une théorie) *Soit P une théorie sous forme clausale et q une clause. Une clause m est dite:*

- impliqué premier *de q relativement à P ssi $P \cup \{q\} \models m$ et pour tout autre clause m', si $P \cup \{q\} \models m'$ et $m' \models m$ alors $m' \equiv m$.*
- impliqué premier propre *de q relativement à P ssi c'est un impliqué premier de q relativement à P mais que $P \not\models m$.*

Définition 4 (Problème du calcul de conséquences) *Soit $\mathcal{P} = (P_i)_{i=1..n}$ une famille de théories sous forme clausale ayant pour langages cibles respectifs $(\mathcal{VC}_{P_i})_{i=1..n}$ et soit $\Gamma = (\mathcal{P}, \text{VOIS})$ un $P2PIS$. Le* problème de calcul de conséquence *consiste, pour un pair P et une clause $q \in \mathcal{L}_P$ donnés, à calculer l'ensemble des impliqués propres de q relativement à $\bigcup_{i=1..n} P_i$, construits à partir de $\mathcal{C}ible(\mathcal{P})$.*

L'exemple qui suit illustre les principales caractéristiques de l'algorithme distribué présenté dans la section 3.

Exemple 1 Considérons quatre pairs P_1,...,P_4. P_1 décrit un tour opérateur. Sa théorie exprime que ses actuelles destinations lointaines (L) sont soit le Kenya (K) soit le Chili (C). Ces destinations lointaines sont des destinations internationales (I) et onéreuses (O). Le pair P_2 s'intéresse aux formalités administratives et exprime qu'un passeport (P) est requis pour les destinations internationales. P_3 s'intéresse aux conditions sanitaires des voyageurs. Il exprime qu'au Kenya, la vaccination contre la fièvre jaune (FV) est fortement recommandée et que de fortes précautions doivent être prises contre le paludisme (PAL) lorsque les logements se font en bungalow (B). P_4 décrit les conditions de logement pour les voyages: bungalow pour le Kenya et hôtels (H) pour le Chili. Il exprime également que lorsque la protection anti-paludisme est requise, les logements sont équipés de protections anti-moustiques (AM). Les variables partagées étiquettent les arêtes du graphe de voisinage (Figure 1) et les variables cibles sont définis par: $\mathcal{VC}_{P_1} = \{O\}$, $\mathcal{VC}_{P_2} = \{P\}$, $\mathcal{VC}_{P_3} = \{B, FJ, PAL\}$ and $\mathcal{VC}_{P_4} = \{B, H, PAL, AM\}$.

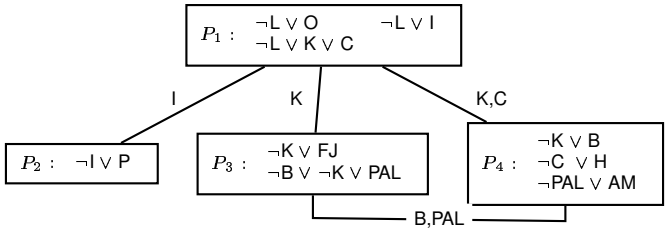

FIG. 1 – *Graphe de voisinage pour l'exemple du tour opérateur*

Supposons que la requête L soit posée au pair P_1. Les conséquences qui peuvent être dérivées par un raisonnement local sur P_1 sont O,I et K ∨ C. O est immédiatement retourné comme une réponse locale puisqu'il est dans $\mathcal{C}ible(P_1)$. Comme I est partagé avec P_2, il est transmis comme requête à P_2, qui produit la clause P. Puisque P est dans $\mathcal{C}ible(P_2)$, elle constitue une réponse distante à la requête initiale. La clause K ∨ C comporte des variables partagées. Notre algorithme découpe de telles clauses et transmet chaque composant (ici K et C) séparément au voisins de P_1 partageant respectivement K et C (avec P_1). C est ainsi transmis à P_4, qui retourne la clause H pour seule réponse. De la même manière, la clause K est transmise (indépendemment) aux pairs P_4 and P_3 (les deux partageant la variable K avec P_1). P_4 produit localement la clause clause B. Comme B ∈ $\mathcal{C}ible(P_4)$ elle est retournée comme réponse pour K. Mais B est également partagée. Elle est donc transmise à P_3, qui produit en retour la clause ¬K ∨ PAL. Cette nouvelle clause est découpée en deux: ¬K et PAL. P_4 est interrogé pour PAL et retourne AM comme seule réponse. P_1 est interrogé pour calculer les impliqués de ¬K cependant, la requête complémentaire K est toujours en cours de traitement. Nous verrons dans la section 3 que ce problème est pris en compte dans notre algorithme grâce à un historique qui garde trace de la branche de raisonnement en cours: lorsqu'une même branche de raisonnement contient deux littéraux opposés, elle est coupée et la clause vide (□) est renvoyée. Dans notre exemple, la réponse produite par P_1 pour

$\neg$K est donc $\Box$, qui est renvoyée en retour à P_3. P_3 peut alors combiner les réponses obtenues par les deux branches de raisonnement résultant du découpage de $\neg$K $\vee$ PAL, à savoir AM, retournée par P_4 pour PAL, et $\Box$, retournée par P_1 pour $\neg$K. Ainsi, P_3 produit AM comme réponse pour $\neg$K $\vee$ PAL. P_3 envoie cette réponse en retour à P_4 en tant que réponse à B. P_4 transmet à P_1 chacune des réponses B, PAL et AM qu'il à obtenues pour L, comme réponses à K. Nous ne détaillerons pas la branche de raisonnement correspondant à la propagation de K dans P_3, qui ajoute une nouvelle réponse, FJ, à l'ensemble des réponses obtenues par P_1 pour K.

Une fois produites, ces réponses sont combinés avec la seule réponse pour C (i.e., H). Ainsi, l'ensemble des réponses produites pour la requête initiale est le suivant :

$\{$H $\vee$ B, H $\vee$ PAL, H $\vee$ AM, H $\vee$ FJ$\}$. Parmi ces réponses, il est important de remarquer que certaines d'entre elles (e.g., H $\vee$ FJ) font intervenir des variables cibles de différents pairs. De tels impliqués ne peuvent être obtenus par des algorithmes fondés sur le partitionnement comme dans [AM]. Ceci est rendu possible ici grâce à la stratégie de découpage/recombinaison mise en œuvre par notre algorithme.

3 Algorithme distribué de calcul de conséquences

Dans la section , nous présentons la version récursive de notre algorithme. Sa version "orientée message" est présenté dans la section 3.2. Après avoir étudié les propriétés de l'algorithme récursif, nous montrerons que l'algorithme orienté message calcul les mêmes résultats que l'algorithme récursif et jouit donc des mêmes propriétés.

La version par passage de message de notre algorithme est décrite dans la section 3.2. Nous montrons qu'il termine et qu'il calcul les même résultats que l'algorithme récursif décrit dans la section 3.1.

Pour les deux algorithmes, nous utiliserons les notations suivantes:

- pour un littéral q, $Resolvant(q, P)$ désigne l'ensemble de clauses obtenues par résolution entre q et une clause de P,
- pour un littéral q, $\bar{q}$ désigne la négation de q,
- pour une clause c d'un pair P, $S(c)$ (resp. $L(c)$) désignent la disjonction des littéraux de c dont les variables sont partagées (resp. non partagées) avec un quelconque voisin de P. Ainsi, la condition $S(c) = \Box$ exprime le fait que c ne partage aucune de ses variables avec ses voisins,
- un historique $hist$ est une liste de triplets (l, P, c) (où l est un littéral, P un pair et c une clause). Un historique $[(l_n, P_n, c_n), \ldots, (l_1, P_1, c_1), (l_0, P_0, c_0)]$ représente une branche de raisonnement engendrée par la propagation du littéral l_0 via le pair P_0 et le découpage de la clause c_0: pour chaque $i \in [0..n-1]$, c_i est une conséquence de l_i et P_i, et l_{i+1} est un littéral de c_i que l'on propage dans P_{i+1},
- $\otimes$ est l'opérateur de distribution sur des ensembles de clauses: $E_1 \otimes \cdots \otimes E_n = \{c_1 \vee \cdots \vee c_n \mid c_1 \in E_1, \ldots, c_n \in E_n\}$. Si $L = \{l_1, \ldots, l_p\}$, on utilise la notation $\otimes_{l \in L} E_l$ pour désigner $E_{l_1} \otimes \cdots \otimes E_{l_p}$.

3.1 Algorithme récursif de calcul de conséquences

Soit $\Gamma = (\mathcal{P}, \text{VOIS})$ un P2PIS. $RCC(q, P, \Gamma)$ calcule les impliqués du littéral q relativement à $\mathcal{P}$, commençant par le calcul des conséquences de q relativement à P, puis se laissant guider par la topologie du graphe de voisinage du P2PIS. Pour garantir la terminaison, il est nécessaire de garder la trace des littéraux déjà calculés par les pairs. Ceci est fait par l'algorithme récursif $RCCH(q, SP, \Gamma, hist)$, avec $hist$ l'historique de la branche de raisonnement terminant par la propagation de q dans SP (un ensemble de voisins du dernier pair ajouté à l'historique).

Algorithme 1: Algorithme récursif de calcul de conséquences
$RCC(q, P, \Gamma)$
(1)**return** $RCCH(q, \{P\}, \Gamma, \emptyset)$

$RCCH(q, SP, \Gamma, hist)$
(1)**si** $(\bar{q}, \_, \_) \in hist$ **return** $\{\Box\}$
(2)**sinon si** il existe $P \in SP$ t.q. $q \in P$ ou pour chaque $P \in SP$, $(q, P, \_) \in hist$ **return** $\emptyset$
(3)**sinon** LOCAL $\leftarrow \{q\} \cup (\bigcup_{P \in SP} Resolvant(q, P))$
(4)**si** $\Box \in$ LOCAL **return** $\{\Box\}$
(5)**sinon** LOCAL $\leftarrow \{c \in \text{LOCAL} | L(c) \in Cible(SP)\}$
(6)**si** pour chaque $c \in$ LOCAL, $S(c) = \Box$, **return** LOCAL
(7)**sinon** RESULTAT $\leftarrow$ LOCAL
(8) **pour tout** $c \in$ LOCAL t.q. $S(c) \neq \Box$
(9) **soit** P le pair de SP t.q. $c \in Resolvant(q, P)$
(10) $\mathcal{P}' \leftarrow \mathcal{P} \backslash \{\neg q \vee c\}$, $\Gamma' \leftarrow (\mathcal{P}', \text{VOIS})$
(11) **pour tout** littéral $l \in S(c)$
(12) REPONSE$(l) \leftarrow RCCH(l, \text{VOIS}(l, P), \Gamma', [(q, P, c)|hist])$
(13) COMBDISJ $\leftarrow (\otimes_{l \in S(c)} \text{REPONSE}(l)) \otimes \{L(c)\}$
(14) RESULTAT $\leftarrow$ RESULTAT $\cup$ COMBDISJ
(15) **return** RESULTAT

Théorème 1 *$RCC(q, P, \Gamma)$ est correct et termine.*

Le théorème suivant énonce une condition suffisante pour que l'algorithme soit complet.

Théorème 2 *Soit $\Gamma = (\mathcal{P}, \text{VOIS})$ un P2PIS dont toutes les théories locales ont été saturées par résolution. Si pour chaque P, P' et $v \in \mathcal{V}_P \cap \mathcal{V}_{P'}$ il existe un chemin entre P et P' dans Γ dont toutes les arêtes sont étiquetées par v, alors pour chaque littéral $q \in \mathcal{L}_P$, $RCC(q, P, \Gamma)$ calcule tout les impliqués propres de q relativement à $\mathcal{P}$ de $Cible(\mathcal{P})$.*

Lemme 1 *Soit P un ensemble de clauses et m un impliqué premier propre de q relativement à P. Soit $P' \subseteq P$, saturée par résolution, telle qu'il contienne les clauses partageant la variable de q. Si m est un impliqué premier propre de q relativement à P, alors:*

- *soit m est un impliqué premier propre de q relativement à P',*
- *soit la variable de q est partagée dans des clauses de $P \backslash P'$,*
- *soit il existe une clause $\neg q \vee c$ de P' tel que c a des variables partagées avec des clauses de $P \backslash P'$ et m est un impliqué premier propre de c relativement à $P \backslash \{\neg q \vee c\} \cup \{q\}$.*

Lemme 2 *Soit P un ensemble de clauses, et soit $c = l_1 \vee \cdots \vee l_n$ une clause. Pour chaque impliqué premier propre m de c relativement à P, il existe $m_1, \ldots, m_n$ tel que $m \equiv m_1 \vee \cdots \vee m_n$, et pour chaque $i \in [1..n]$, m_i est un impliqué premier propre de l_i relativement à P.*

3.2 Algorithme "orienté message" de calcul de conséquences

Dans cette section, nous présentons le résultat de la transformation du précédent algorithme en une version pae passage de message. L'algorithme implanté localement dans chaque pair, est composé de trois procédures. Chacune d'elle peut être déclenchée par l'arrivée d'un message.

La procédure RECEPTIONMESSAGEREQUETE est déclenchée par la réception d'un message $requete\ m(Exp, Dest, requete, hist, l)$ envoyé par le pair Exp au pair $Dest$ qui exécute la procédure: à la demande de Exp, avec qui il partage la variable de l, il traite le littéral l.

La procédure RECEPTIONMESSAGEREPONSE est déclenchée par la réception d'un message $reponse\ m(Exp, Dest, reponse, hist, r)$, envoyé par le pair Exp au pair $Dest$ qui exécute la procédure: il traite la réponse r (qui est une clause dont les variables sont des variables cibles) envoyé par Exp pour le littéral l (qui est le dernier à avoir été ajouté dans l'historique) ; cette réponse doit être combinée avec les réponses des autres littéraux apparaissant dans la même clause que l.

La procédure RECEPTIONMESSAGEFINAL est déclenchée par la réception d'un message $final\ m(Exp, Dest, final, hist, vrai)$: le pair Exp avertit le pair $Dest$ que le calcul des réponses pour le littéral l (le dernier à avoir été ajouté dans l'historique) est terminé.

Ces procédures exploitent deux structures de données stockées localement sur chaque pair : REPONSE$(l, hist)$ stocke les réponses issues de la propagation de l à travers la branche de raisonnement correspondant à $hist$; FINAL$(q, hist)$ vaut vrai quand la propagation de q à travers la branche de raisonnement correspondant à $hist$ est terminée. Le raisonnement est amorcé par l'utilisateur (que l'on considère comme un pair particulier appelé $Utilisateur$) envoyant à un pair donné P le message $m(Utilisateur, P, requete, \emptyset, q)$, qui déclenche la procédure RECEPTIONMESSAGEREQUETE$(m(Utilisateur, P, requete, \emptyset, q))$ qui est exécutée localement par P. Les procédures, étant exécutées localement par le pair recevant un message, dans leur description nous noterons Moi le pair destinataire du message.

Le théorème suivant énonce deux résultats importants: premièrement, l'algorithme distribué par passage de message calcule les même résultats que l'algorithme de la section 3.1. Il est donc complet sous les mêmes conditions que dans le théorème 2. Deuxièmement, l'utilisateur est avertit de la terminaison quand elle a lieu, ce qui est primordial dans le cadre d'un algorithme anytime.

Théorème 3 *Soit r le résultat retourné par $RCC(q, P, \Gamma)$. Si P reçoit de l'utilisateur le message $m(Utilisateur, P, requete, \emptyset, q)$, alors le message $m(P, Utilisateur, reponse, [(q, P, \_)], r)$ sera emis. Si r est le dernier resultat retourné par $RCC(q, P, \Gamma)$, alors l'utilisateur sera avertit de la terminaison par un message $m(P, Utilisateur, final, [(q, P, vrai)], vrai)$.*

Pour simplifier, les algorithmes que nous avons présentés s'appliquent à des requête réduites à des littéraux. On peut cependant facilement les étendre pour des clauses quelconques: il suffit pour cela de découper ces clauses en littéraux et d'utiliser l'opérateur $\veebar$ pour recombiner les résultats obtenus pour chaque littéral.

4 Expérimentations

Une architecture P2PIS a été développé en Java et déployée sur un cluster de 28 machines Linux Athlon 1800+ avec 1GB de mémoire. Le calcul des impliqués locaux est assuré par une

Algorithme 2: Procédure pour le traitement d'un message requête
RECEPTIONMESSAGEREQUETE($m(Exp, Moi, requete, hist, q)$)
(1) **si** $(\bar{q}, \_, \_) \in hist$
(2) **envoie** $m(Moi, Exp, reponse, [(q, Moi, \Box)|hist], \Box)$
(3) **envoie** $m(Moi, Exp, final, [(q, Moi, vrai)|hist], vrai)$
(4)**sinon si** $q \in Moi$ ou $(q, Moi, \_) \in hist$
(5) **envoie** $m(Moi, Exp, final, [(q, Moi, vrai)|hist], vrai)$
(6)**sinon**
(7) LOCAL$(Moi) \leftarrow \{q\} \cup Resolvant(q, Moi)$
(8) **si** $\Box \in$ LOCAL(Moi)
(9) **envoie** $m(Moi, Exp, reponse, [(q, Moi, \Box)|hist], \Box)$
(10) **envoie** $m(Moi, Exp, final, [(q, Moi, vrai)|hist], vrai)$
(11) **sinon**
(12) LOCAL$(Moi) \leftarrow \{c \in$ LOCAL$(Moi)|\ L(c) \in Cible(Moi)\}$
(13) **si** pour chaque $c \in$ LOCAL$(Moi), S(c) = \Box$
(14) **pour tout** $c \in$ LOCAL(Moi)
(15) **envoie** $m(Moi, Exp, reponse, [(q, Moi, c)|hist], c)$
(16) **envoie** $m(Moi, Exp, final, [(q, Moi, vrai)|hist], vrai)$
(17) **sinon**
(18) **pour tout** $c \in$ LOCAL(Moi)
(19) **si** $S(c) = \Box$
(20) **envoie** $m(Moi, Exp, reponse, [(q, Moi, c)|hist], c)$
(21) **sinon**
(22) **pour tout** littéral $l \in S(c)$
(23) **si** $l \in Cible(Moi)$
(24) REPONSE$(l, [(q, Moi, c)|hist]) \leftarrow \{l\}$
(25) **sinon**
(26) REPONSE$(l, [(q, Moi, c)|hist]) \leftarrow \emptyset$
(27) FINAL$(l, [(q, Moi, c)|hist]) \leftarrow faux$
(28) **pour tout** $PD \in$ VOIS(l, Moi)
(29) **envoie** $m(Moi, PD, requete, [(q, Moi, c)|hist], l)$

Algorithme 3: Procédure pour le traitement d'un message réponse
RECEPTIONMESSAGEREPONSE($m(Exp, Moi, reponse, hist, r)$)
(1)$hist$ est de la forme $[(l', Exp, c'), (q, Moi, c)|hist']$
(2)REPONSE$(l', hist) \leftarrow$ REPONSE $(l', hist) \cup \{r\}$
(3)RESULTAT$\leftarrow \Bbbwedge_{l \in S(c) \setminus \{l'\}}$REPONSE$(l, hist) \Bbbwedge \{L(c) \vee r\}$
(4)**si** $hist' = \emptyset, U \leftarrow Utilisateur$ **sinon** $U \leftarrow$ le premier pair P' de $hist'$
(5)**pour tout** $cs \in$ RESULTAT
(6) **envoie** $m(Moi, U, reponse, [(q, Moi, c)|hist'], cs)$

version locale de l'algorithme distribué que nous avons présenté; nous avons optimisé le comportement global du programme en éliminant autant que possible les clauses subsumées (les éliminer toutes étant impossible compte tenu du caractère anytime et distribué de notre algorithme).

Génération de benchmark: Évaluer les performances d'un système distribué n'est pas trivial. Nous nous sommes concentrés sur des théories aléatoires car elles ont été largement étudiées (dans un contexte centralisé) dans la littérature ([SC96]). Elles représentent également un véritable

Algorithme 4: Procédure pour le traitement d'un message final
RECEPTIONMESSAGEFINAL($m(Exp, Moi, final, hist, vrai)$)
(1)$hist$ est de la forme $[(l', Exp, vrai), (q, Moi, c)|hist']$
(2)FINAL$(l', hist) \leftarrow vrai$
(3)**si** pour chaque $l \in S(c)$, FINAL$(l, hist) = vrai$
(4) **si** $hist' = \emptyset$ $U \leftarrow Utilisateur$ **sinon** $U \leftarrow$ le premier pair P' de $hist'$
(5) **envoie** $m(Moi, U, final, [(q, Moi, vrai)|hist'], vrai)$
(6) **pour tout** $l \in S(c)$
(7) REPONSE$(l, [(l, Exp, _), (q, Moi, c)|hist']) \leftarrow \emptyset$

challenge pour la compilation : de petites théories peuvent engendrer d'énormes théories compilées. Notre générateur de benchmark prend en entrée les caractéristiques de notre graphe de voisinage (d noeuds et e arêtes). Chaque noeud contient une théorie en 3CNF générée aléatoirement, contenant un nombre fixe de m clauses et de n variables. Chaque arête du graphe connexe généré est étiqueté par q variables qui sont nécessairement communes aux deux pairs concernés. Afin d'encoder simplement une variable partagée tout en s'assurant que le théorème 2 est applicable, nous ajoutons deux clauses dans les deux pairs concernés pour assurer l'équivalence entre la variable locale et distante. Ainsi, on peut affirmer que la théorie globale sous-jacente contient $d.m$ clauses aléatoires de taille 3, $d.n$ variables, et $4.q.e$ clauses de taille 2 qui encodent l'équivalence entre variables partagées. Un autre paramètre p caractérise le nombre de variables cibles de chaque pair.

Dans notre expérience, nous avons fixé $q = 1$ et $e = 1.3 * d$ (pour ne pas obtenir de graphes trop contraints), avec $d \in \{5, 10, 28\}$. Nous avons limité nos tests à de petits pairs (contenant moins de 30 clauses). De telles théories peuvent déjà contenir un nombre important d'impliqués [SC96]. Les algorithmes les plus performants [SdV] atteignent leurs limites sur des théories aléatoires pouvant aller jusqu'à 150 clauses et 50 variables, ce qui correspond à seulement 5 pairs de 30 clauses.

Analyse expérimentale: Notre première constatation est que, de manière générale, les clauses les plus courtes apparaissent en premier et que les clauses produites contiennent des variables provenant de plusieurs pairs distants. Nous avons également observé des différences significatives dans le comportement de l'algorithme pour différentes requêtes posées aux mêmes P2PIS. Nous imposons une limite de temps de 30s pour chaque requête, qui représente un temps d'attente limite raisonnable pour un utilisateur interrogeant un P2PIS.

Les valeurs moyennes obtenues sont présentés dans le tableau 1. Chaque benchmark consiste en une synthèse effectuée sur un nombre $\#Q$ de requêtes différentes sur le même P2PIS. La colonne $\#Q$ donne le nombre de requêtes différentes posées au même P2PIS, le nombre de requêtes qui ont terminé avant la limite de temps et leurs temps de calcul

La colonne #Imp donne le nombre d'impliqués trouvés et sa médiane. La dernière colonne donne le temps qu'il a fallu pour produire respectivement 2, 10, 100 et 1000 réponses (quant elle existes).Ces valeurs donnent une idée de la vitesse de production de l'algorithme.

Les deux premières lignes ($d = 5$) montrent que réduire la taille du langage cible n'augmente pas nécessairement les performances. Pour un langage cible réduit, les pairs interrogeront moins leur voisinage et pourront trouver rapidement toutes les réponses (19 requêtes ont terminées en 5s). D'autre part, plus le langage cible est large, plus les premières réponses apparaîtront rapidement.

Pour les expériences avec 10 pairs, on observe de grosses différences pour la colonne #Imp. Ceci peut être expliqué par une distribution peu homogène favorisant de nombreuses différences

d	m	n	p	#Q (#Terminé)	#Imp	Temps
5	22	11	2	55 (19, 5s)	14 (7)	2,8,–,–
5	22	11	5	55 (3, 1s)	7799 (1431)	1,3,5,8
10	22	11	5	40 (4, 1s)	23132 (1651)	1,2,5,9
10	30	15	5	40 (4, 2s)	7099 (648)	3,7,13,17
28	22	11	5	112 (20, 2s)	8120 (513)	2,3,6,10
28	30	15	5	112 (7, 4s)	1132 (54)	8,13,20,24

TAB. 1 – *Résultats sur de nombreuses requêtes posées à différents P2PIS*

de comportement d'un même P2PIS sur différentes requêtes. On peut voir que même si seulement 10% des requetes terminent, la vitesse de production de l'algorithme est bonne (1000 clauses produites en moins de 10s pour $m = 22$). Quand la complexité des théories locales augmente ($m = 30$), l'impact ressenti est la diminution de la vitesse de production.

Enfin, nous avons testé le passage à l'échelle de notre approche en déployant l'architecture sur les 28 noeuds du cluster, avec des tailles de théories différentes.

Même si peu de requêtes terminent avant la limite de temps, notre algorithme passe bien à l'échelle, et produit rapidement un assez grand nombre de clauses. On remarque que lorsque les théories locales deviennent difficiles, les premiers résultats arrivent raisonnablement vite. Notons que sur de telles théories (représentant 988 clauses et 420 variables si on fait l'union de toutes les théories locales) `zres` [SdV] n'a toujours pas terminé son calcul au bout de plus d'une heure.

5 Travaux connexes

L'algorithme décrit dans la section 3 peut être vu comme une version distribuée d'un algorithme de déduction de clauses d'un langage cible par résolution linéaire ordonnée [CL73], produisant de nouvelles clauses d'un langage cible, technique reprise par Siegel dans [Sie87] (et étendue au premier ordre par Inoue [Ino92]).

Le calcul de nouvelles clauses dérivées (les "impliqués propres" dans la section 2) à été largement étudié (voir [Mar99] pour un résumé). En particulier, ce problème, également connu sous le nom de calcul de Φ-impliqués a été étudié dans [Sie87, Ino92] et dans [KT90].

Nous avons souligné les différences entre nos travaux et ceux de [AM]. Dans un environnement pair-à-pair, utiliser la décomposition en arbre du graphe de voisinage est impossible, mais nous pouvons appliquer notre algorithme à des théories partitionnées à la place de celui de [AM]. Il pourrait tirer partie de la décomposition en arbre pour améliorer les performances. Comme nous l'avons montré dans l'exemple introductif, l'algorithme de [AM] exige pour être complet que le langage cible global soit l'union des langages cibles locaux. [GR03] exploite cette particularité dans le but d'encoder un P2PIS ayant une architecture pair/super-pairs en une théorie propositionnelle partitionnée et ainsi utiliser l'algorithme de calcul de conséquences de [AM]. La connaissance globale des variables cibles de tout le PDMS doit être connu puis distribuée à travers les supers pairs. L'environnement de diagnostic pour des systèmes embarqués distribués ([Pro]) est fondé sur [AM]. Nous pensons qu'il peut bénéficier de notre approche pour être appliqué à un véritable environnement pair-à-pair dans lequel aucune connaissance globale ne peut être partagée.

Dans les ATMS distribués [MJ89], des agents échangent des ensembles de nogoods dans le but de converger vers un ensemble globalement consistant de justifications. À la différence de la vision pair-à-pair, une telle vision des ATMS est lié à une connaissance globale partagée par tout

les agents et à pour objectif de converger vers une unique solution globale.

6 Conclusion et perspectives

Les contributions de cet article sont à la fois théoriques et pratiques. Nous proposons le premier algorithme distribué de calcul de conséquences dans un environnement pair-à-pair, et nous énonçons une condition suffisante pour garantir sa complétude. Nous avons développé une architecture P2PIS qui implémente l'algorithme et pour laquelle nous obtenons de premiers résultats prometteurs. Cette architecture est utilisée dans un projet en collaboration avec France Télécom, qui a pour objectif d'enrichir les applications web de type pair-à-pair avec des services de raisonnement (e.g., Someone [PBAvdV03]).

Jusqu'ici, nous avons restreint notre algorithme à l'utilisation d'un langage cible fondé sur le vocabulaire. Cependant, il peut être adapté à des langages cibles plus sophistiqués (e.g., impliqués d'une taille maximale donnée, langages fondés sur les littéraux et non pas que sur les variables). Ceci peut être fait en ajoutant une simple balise sur tous les messages pour encoder le langage cible désiré. Une autre extension possible de notre algorithme serait d'autoriser une représentation plus compacte des impliqués, comme cela est fait dans [SdV]. Ces travaux sont liés à un opérateur efficace de distribution sur les clauses. Cette stratégie peut être adaptée en étendant les messages de notre algorithme dans l'objectif d'envoyer des ensembles *compressés* de clauses à la place d'une seule clause comme c'est le cas actuellement, et ceci sans changer profondément le principe de notre algorithme. Nous prévoyons également de rendre notre implémentation plus robuste et dynamique en utilisant une couche réseau de type Chord [SMK$^+$] dans l'objectif de gérer les problèmes de connections/déconnections des pairs à tout instant.

Remerciements

Nous remercions P. Siegel pour les fructueuses discussions concernant les algorithmes de productions et les relecteurs anonymes pour la pertinence de leurs remarques et suggestions.

Références

[AM] E. Amir and S. McIlraith. Partition-based logical reasoning. In *KR'00*.

[CL73] C. L. Chang and R. C. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Computer Science Classics, Academic Press, 1973.

[DR] R. Dechter and I. Rish. Directed resolution: the davis-putnam procedure revisited. In *KR'94*.

[dV] A. del Val. A new method for consequence finding and compilation in restricted languages. In *AAAI'99*.

[Gnu] Gnutella. http://gnutella.wego.com.

[GR03] F. Goasdoue and M.-C. Rousset. Querying distributed data through distributed ontologies: a simple but scalable approach. *IEEE Intelligent Systems*, (18), 2003.

[HIST] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *ICDE'03*.

[HITM] A. Halevy, Z. Ives, I. Tatarinov, and Peter Mork. Piazza: data management infrastructure for semantic web applications. In *WWW'03*.

[Ino92] K. Inoue. Linear resolution for consequence finding. *Artificial Intelligence*, (56), 1992.

[KT90] A. Kean and G. Tsiknis. An incremental method for generating prime implicants/implicates. *Journal of Symbolic Computation*, 9, 1990.

[Mar99] P. Marquis. *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 5, chapter Consequence Finding Algorithms. Kluwer Academic Publishers, 1999.

[MJ89] C.L. Mason and R.R. Johnson. *Distributed Artificial Intelligence II*, chapter DATMS: a framework for distributed assumption based reasoning. Pitman, 1989.

[Nap] Napster. http://www.napster.com.

[NWQ+] W. Nedjl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch. Edutella: a p2p networking infrastructure based on rdf. In *WWW'02*.

[PBAvdV03] M. Plu, P. Bellec, L. Agosto, and W. van de Velde. The web of people: A dual view on the WWW. In *Int. World Wide Web Conf.*, 2003.

[Pro] G. Provan. A model-based diagnosis framework for distributed embedded systems. In *KR'02*.

[SC96] R. Schrag and J. Crawford. Implicates and prime implicates in random 3-sat. *Artificial Intelligence*, 81, 1996.

[SdV] L. Simon and A. del Val. Efficient consequence finding. In *IJCAI'01*.

[Sie87] P. Siegel. *Représentation et utilisation de la connaissance en calcul propositionnel*. PhD thesis, Université d'Aix-Marseille II, 1987.

[SMK+] I. Stoica, R. Morris, D. Karger, M.F. Kaasshoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. In *2001 Conference on applications, technologies, archtecture and protocols for computer communications*.