Scalability Study of Peer-to-Peer Consequence Finding

P. Adjiman and **P. Chatalic** and **F. Goasdoué** and **M.-C. Rousset** and **L. Simon**
PCRI: CNRS & Université Paris-Sud (LRI) – INRIA (Futurs)
Bât. 490, 91405, Université Paris-Sud, Orsay Cedex, France
{adjiman,chatalic,fg,mcr,simon}@lri.fr

Abstract

In peer-to-peer inference systems, each peer can reason locally but also solicit some of its acquaintances, sharing part of its vocabulary. This paper studies both theoretically and experimentally the problem of computing proper prime implicates for propositional peer-to-peer systems, the global theory (union of all peer theories) of which is not known (as opposed to partition-based reasoning).

1 Introduction

Recently peer-to-peer (P2P) systems have received considerable attention because their underlying infrastructure is appropriate to scalable and flexible distributed applications over Internet. In a full P2P system, there is no centralized control or hierarchical organization: each peer is equivalent in functionality and cooperates with other peers in order to solve a collective task. First P2P systems were simple keyword-based file sharing systems like Napster (http://www.napster.com) and Gnutella (http://gnutella.wego.com), for which efficient lookup services (e.g., CHORD [Stoica *et al.*, 2001]) have been invented. Recently, schema-based peer data management systems like Edutella [Nedjl *et al.*, 2002] or Piazza [Halevy *et al.*, 2003b] have been proposed. In those systems, query answering complexity is directly related to the expressivity of the formalism used to state the semantic mappings between peers schemas [Halevy *et al.*, 2003a]. The scalability of Piazza so far goes up to about 80 peers and relies on a wide range of optimizations (mappings composition, paths pruning [Tatarinov and Halevy, 2004]), made possible by the centralized storage of all the schemas and mappings in a global server.

In this paper, we make the choice of being fully distributed: there are neither super-peers (as in Edutella) nor a central server (as in Piazza). In addition, we aim at scaling up to thousands of peers. We consider P2P inference systems in which the local theory of each peer is a set of clauses defined upon a set of propositional variables. Each peer may share part of its vocabulary with some other peers. We investigate the reasoning task of finding consequences of a certain form for a given input formula expressed using the local vocabulary of a peer. This reasoning task is important in many applications (diagnosis, information integration), in which output must be computed from input that is provided at query time, and cannot be reduced to satisfiability checking.

The problem of distributed reasoning considered in this paper is quite different from the problem of reasoning over partitions obtained by decomposition of a theory ([Dechter and Rish, 1994; Amir and McIlraith, 2000]). In that problem, a centralized large theory is given and its structure is exploited to compute its best partitioning, in order to optimize the use of a partition-based reasoning algorithm. In our problem, the whole theory (i.e., the union of all the local theories) is not known and the partition is imposed by the P2P architecture. Therefore, existing algorithms ([Amir and McIlraith, 2000; Dechter and Rish, 1994; del Val, 1999]) are not appropriate for our consequence finding problem.

Section 2 defines formally the P2P inference problem addressed in this paper. Section 3 describes the distributed consequence finding algorithm that we propose and states its properties. Section 4 accounts for a significant experimental study of the scalability of this approach. Section 5 describes related work and we conclude in Section 6.

2 Peer-to-peer inference: problem definition

A peer-to-peer inference system (P2PIS) is a network of peer theories. Each peer P is a finite set of formulas of a language $\mathcal{L}_P$. We consider the case where $\mathcal{L}_P$ is the language of clauses that can be built from a finite set of propositional variables $\mathcal{V}_P$, called the *vocabulary* of P. Peers can be semantically related by having common *shared variables* in their respective vocabularies. Each peer only knows its own local theory and that it shares some variables with some other peers of the P2PIS (its *acquaintances*). It does not necessarily know *all* the peers with which it shares variables. When a new peer joins a P2PIS it simply declares its acquaintances in the network, i.e., the peers it knows to be sharing variables with. A P2PIS can be formalized as an *acquaintance graph*.

Definition 1 *Let $\mathcal{P} = \{P_i\}_{i=1..n}$ be a collection of clausal theories on their respective vocabularies $\mathcal{V}_{P_i}$, let $\mathcal{V} = \cup_{i=1..n}\mathcal{V}_{P_i}$. An* ***acquaintance graph*** *over $\mathcal{V}$ is a graph $\Gamma = (\mathcal{P}, \text{ACQ})$ where $\mathcal{P}$ is the set of vertices and $\text{ACQ} \subseteq \mathcal{V} \times \mathcal{P} \times \mathcal{P}$ is a set of labelled edges such that for every $(v, P_i, P_j) \in \text{ACQ}$, $i \neq j$ and $v \in \mathcal{V}_{P_i} \cap \mathcal{V}_{P_j}$.*

A labelled edge (v, P_i, P_j) expresses that peers P_i and P_j know each other to be sharing the variable v. For a peer P and a literal l, $\text{ACQ}(l, P)$ denotes the set of peers sharing with P

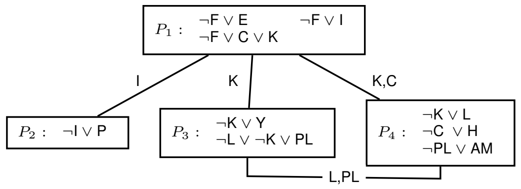

Figure 1: Acquaintance graph for the tour operator example

the variable of l. For each theory P, we consider a subset of *target variables* $\mathcal{TV}_P \subseteq \mathcal{V}_P$, supposed to represent the variables of interest for the application. The goal is, given a clause provided as an input to a given peer, to find all the possible consequences of the input clause and the union of the peer theories, that belong to some *target language*. Given a set SP of peers, the target language $\mathcal{T}arget(SP)$ is the language of clauses (including the empty clause) involving only target variables of peers of SP. A shared variable must have the same target status in all the peers sharing it.

Definition 2 *Let P be a clausal theory and q be a clause. A clause m is called a **prime implicate** of q **w.r.t.** P iff $P \cup \{q\} \models m$ and for any other clause m', if $P \cup \{q\} \models m'$ and $m' \models m$ then $m' \equiv m$. m is called a **proper prime implicate** of q **w.r.t.** P iff it is a prime implicate of q w.r.t. P but $P \not\models m$.*

Definition 3 *Let $\Gamma = (\mathcal{P}, \text{ACQ})$ be an acquaintance graph modeling a $P2PIS$, where $\mathcal{P} = \{P_i\}_{i=1..n}$ is a collection of clausal theories with respective target variables. The **consequence finding problem** is, given a peer P and a clause $q \in \mathcal{L}_P$ to find the set of proper prime implicates of q w.r.t. $\bigcup_{i=1..n} P_i$ which belong to $\mathcal{T}arget(\mathcal{P})$.*

Since none of the peers of a P2PIS knows the union of the theories of the system, the consequence finding problem in a P2PIS is new and significantly different from the consequence finding problem in a single global theory (even partitioned). In a full P2P setting, each peer must run the same reasoning algorithm locally and independently, while being able to distribute part of the reasoning task that it controls to some of its acquaintances.

Example : Let us consider 4 peers (Figure 1). P_1, describing a tour operator, expresses that its current far destinations (F) are either Kenya (K) or Chile (C). These far destinations are international destinations (I) and expensive (E). P_2, only concerned with police regulations, expresses that a passport is required (P) for international destinations. P_3 focuses on sanitary conditions for travelers. It expresses that, in Kenya, yellow fever vaccination (Y) is strongly recommended and that a strong protection against paludism should be taken (PL) when accomodation occurs in Lodges (L). P_4 describes accommodation conditions : Lodges for Kenya and Hotels (H) for Chile. It also expresses that when anti-paludism protection is required, accommodations are equipped with appropriate anti-mosquito protections (AM). Shared variables are indicated on the edges of the acquaintance graph (Figure 1) and target variables are defined by : $\mathcal{TV}_{P_1} = \{E\}$, $\mathcal{TV}_{P_2} = \{P\}$, $\mathcal{TV}_{P_3} = \{L, Y, PL\}$ and $\mathcal{TV}_{P_4} = \{L, H, PL, AM\}$.

We now illustrate the behavior of the distributed consequence finding algorithm detailed in Section 3. When a peer receives a query, it first computes all local consequents of the query. Those in $\mathcal{T}arget(P)$ are immediately returned. Then, those made of shared literals are splitted. For each shared literal, a subquery is propagated to the neighbor peers sharing the corresponding variable. When returned, consequents of the subqueries are respectively queued for future recombination. As soon as one answer has been returned for each subquery, they are recombined and transmitted back as new consequents to the querying peer. This process continues incrementally, as further consequents for the subqueries are returned.

For instance, suppose that F is transmitted to peer P_1 by the user. The consequents that are locally computed are E, I and $\mathsf{C} \vee \mathsf{K}$. Since $\mathsf{E} \in \mathcal{T}arget(P_1)$, it is immediately returned as a *local consequent*. Since I is shared with P_2, it is transmitted to P_2, which produces the clause P. Since $\mathsf{P} \in \mathcal{T}arget(P_2)$, it is transmitted back to P_1 and returned as a *remote consequent* of the initial query. The clause $\mathsf{C} \vee \mathsf{K}$, being made of shared variables, is splitted and C and K are transmitted separately to the concerned neighbors. C is transmitted to P_4, which returns (only) H to P_1, where it is queued for combination. Similarly, K is transmitted (independently) to P_4 and P_3 (both share K with P_1). On P_4, L is produced locally. Since $\mathsf{L} \in \mathcal{T}arget(P_4)$ it is returned as a first consequent of K to P_1, where it is queued. On P_1, after recombination, $\mathsf{H} \vee \mathsf{L}$ is then returned as a first *combined consequents* of the initial query. Since L is also shared between P_4 and P_3, it is propagated on P_3, where the clause $\neg\mathsf{K} \vee \mathsf{PL}$ is produced and, in turn, splitted. P_4 is then asked for PL and returns AM as its only consequent. P_1 is asked for $\neg\mathsf{K}$. This happens while the complementary query K is still under process. We will see in Section 3 that when a same reasoning branch contains two complementary literals (which is detected using a history mechanism), it is closed and $\Box$ is returned as a consequent. P_3 now combines AM (returned by P_4 for PL), and $\Box$ (returned by P_1 for $\neg\mathsf{K}$) as a new consequent of $\neg\mathsf{K} \vee \mathsf{PL}$, and thus, transmits AM back to P_4 as a new consequent of L. For lack of space, we do not detail all reasoning branches. The set of consequents of the initial query eventually produced is: $\{\mathsf{E}, \mathsf{I}, \mathsf{H} \vee \mathsf{L}, \mathsf{H} \vee \mathsf{AM}, \mathsf{H} \vee \mathsf{Y}, \mathsf{H} \vee \mathsf{PL}\}$. Among those answers, let us note that some of them (e.g., $\mathsf{H} \vee \mathsf{Y}$) involve target variables from different peers. Such implicates cannot be obtained by partition-based algorithms like in [Amir and McIlraith, 2000] which only compute consequents that belong to the target language of a single peer.

3 Distributed consequence finding algorithm

Our distributed and anytime consequence finding algorithm is has been presented in [Adjiman *et al.*, 2004b]. For this paper to be self-contained, we describe the three main message passing procedures, which are implemented locally at each peer. They are triggered by the reception of a $query$ (resp. $answer$, $final$) message, sent by a $Sender$ peer to a receiver peer, denoted by $Self$, which executes the procedure.

Those procedures handle an *history* which is initialized to the empty sequence. An history $hist$ is a sequence of triples (l, P, c) (where l is a literal, P a peer, and c a clause). An history $[(l_n, P_n, c_n), \ldots, (l_1, P_1, c_1), (l_0, P_0, c_0)]$ represents a branch of reasoning initiated by the propagation of the literal l_0 within the peer P_0, and the splitting of the clause c_0: for every $i \in [0..n-1]$, c_i is a consequence of l_i and P_i, and l_{i+1} is a literal of c_i, which is propagated in P_{i+1}.

The reasoning is initiated by the user (denoted by a particular peer $User$) sending to a given peer P a message $m(User, P, query, \emptyset, q)$, which triggers the procedure ReceiveQueryMessage$(m(User, P, query, \emptyset, q))$ that is locally executed by P. In the description of the procedures, we will use the notations:

• for a literal q, $\bar{q}$ denotes its complementary literal and $Resolvent(q, P)$ denotes the set of clauses obtained by resolution between q and a clause of P,

• for a clause c of a peer P, $S(c)$ (resp. $L(c)$) denotes the disjonction of literals of c whose variables are shared (resp. not shared) with any acquaintance of P. $S(c) = \square$ thus expresses that c does not contain any shared variable,

• $\circledvee$ is the distribution operator on sets of clauses: $S_1 \circledvee \cdots \circledvee S_n = \{c_1 \vee \cdots \vee c_n \mid c_1 \in S_1, \ldots, c_n \in S_n\}$. If $L = \{l_1, \ldots, l_p\}$, we use $\circledvee_{l \in L} S_l$ to denote $S_{l_1} \circledvee \cdots \circledvee S_{l_p}$.

The main properties of the resulting distributed message passing algorithm, stated in [Adjiman *et al.*, 2004a], can be summarized as follows:

1) The algorithm is sound: the answers that are returned are all implicates of the literal having triggered the reasoning.

2) The algorithm terminates and the user is notified of the termination, which is crucial for an anytime algorithm.

3) The completeness of the algorithm is guaranteed if each local theory is saturated by resolution and if the acquaintance graph is such that if two local theories have a common variable, there exists in the acquaintance graph a path between those two theories, all the edges of which are labeled with that variable. If that property is not satisfied, the algorithm still applies but does not guarantee to return all the proper prime implicates when it terminates.

Note that $\square$ can be returned by our algorithm (lines 1 to 3 and 8 to 10 in ReceiveQueryMessage). Therefore, our algorithm can be exploited for checking the satisfiability of the global theory at each join of a new peer. For the sake of simplicity, our algorithm is presented as applying to literals. However any clause can be handled by splitting it into literals and recombining the results obtained for each literal using the $\circledvee$ operator.

Algorithm 1: Procedure handling queries. It takes care of the propagation of the literal q by $Self$.

```
ReceiveQueryMessage($m(Sender, Self, query, hist, q)$)
(1) if $(\bar{q}, \_, \_) \in hist$
(2)   send $m(Self, Sender, answer, [(q, Self, \square)|hist], \square)$
(3)   send $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(4) else if $q \in Self$ or $(q, Self, \_) \in hist$
(5)   send $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(6) else
(7)   LOCAL($Self$) $\leftarrow \{q\} \cup Resolvent(q, Self)$
(8)   if $\square \in$ LOCAL($Self$)
(9)     send $m(Self, Sender, answer, [(q, Self, \square)|hist], \square)$
(10)    send $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(11)  else
(12)    LOCAL($Self$) $\leftarrow \{c \in$ LOCAL($Self$)$|\ L(c) \in \mathcal{T}arget(Self)\}$
(13)    if for every $c \in$ LOCAL($Self$), $S(c) = \square$
(14)      foreach $c \in$ LOCAL($Self$)
(15)        send $m(Self, Sender, answer, [(q, Self, c)|hist], c)$
(16)      send $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(17)    else
(18)      foreach $c \in$ LOCAL($Self$)
(19)        if $S(c) = \square$
(20)          send $m(Self, Sender, answer, [(q, Self, c)|hist], c)$
(21)        else
(22)          foreach literal $l \in S(c)$
(23)            if $l \in \mathcal{T}arget(Self)$
(24)              ANSWER($l, [(q, Self, c)|hist]$) $\leftarrow \{l\}$
(25)            else
(26)              ANSWER($l, [(q, Self, c)|hist]$) $\leftarrow \emptyset$
(27)            FINAL($l, [(q, Self, c)|hist]$) $\leftarrow false$
(28)            foreach $RP \in$ ACQ($l, Self$)
(29)              send $m(Self, RP, query, [(q, Self, c)|hist], l)$
```

Algorithm 2: Procedure handling answers. r is returned as a consequent of the last literal added to the history $hist$.

```
ReceiveAnswerMessage($m(Sender, Self, answer, hist, r)$)
(1) $hist$ is of the form $[(l', Sender, c'), (q, Self, c)|hist']$
(2) ANSWER($l', hist$) $\leftarrow$ ANSWER ($l', hist$) $\cup \{r\}$
(3) RESULT$\leftarrow \circledvee_{l \in S(c)\setminus\{l'\}}$ANSWER($l, hist$) $\circledvee \{L(c) \vee r\}$
(4) if $hist' = \emptyset$, $U \leftarrow User$ else $U \leftarrow$ the first peer $P'$ of $hist'$
(5) foreach $cs \in$ RESULT
(6)   send $m(Self, U, answer, [(q, Self, c)|hist'], cs)$
```

Algorithm 3: Procedure handling notifications: answer computation for the last literal added to $hist$ is completed.

```
ReceiveFinalMessage($m(Sender, Self, final, hist, true)$)
(1) $hist$ is of the form $[(l', Sender, true), (q, Self, c)|hist']$
(2) FINAL($l', hist$) $\leftarrow true$
(3) if for every $l \in S(c)$, FINAL($l, hist$) $= true$
(4)   if $hist' = \emptyset$ $U \leftarrow User$ else $U \leftarrow$ the first peer $P'$ of $hist'$
(5)   send $m(Self, U, final, [(q, Self, true)|hist'], true)$
(6)   foreach $l \in S(c)$
(7)     ANSWER($l, [(l, Sender, \_), (q, Self, c)|hist']$) $\leftarrow \emptyset$
```

4 Experimental analysis

In order to study scalability issues of our P2P algorithm we have conducted a significant experimentation on networks composed of 1000 peers. To the best of our knowledge, this is the first experimental study on such large P2PIS. Our motivation was twofold. First, to study how deep and how wide reasoning spreads on the network. Second, to evaluate the time needed to obtain answers and to check to what extent the P2PIS is able to support the traffic load.

Since we want to use our infrastructure in real Web applications, we have chosen to generate acquaintance graphs having the so-called *small world* property, which is admitted [Newman, 2000] as being a general property of social networks (including the Web). Following [Watts and Strogatz, 1998], we start from a regular ring of 1000 nodes, 10 edges per node, and rewire each node with a given probability pr. It has been shown that between regular graphs ($pr = 0$) and uniform random graphs ($pr = 1$), the graphs generated with $pr = 0.1$ have "small world" properties.

In the following experiments, the number q of shared variables labelling each edge varies, and each of the 1000 local theories is a 2+p clausal theory composed of clauses of length 2 and a varying ratio p of clauses of length 3. Each local theory is generated in two steps. First, 70 clauses of length 2 are uniformly generated over 70 variables that are proper to the local theory, among which 40 are chosen as target variables. Then, we add clauses, involving shared variables, of length 2 or 3. We denote $\%3cnf$ the percentage of clauses of length 3 to generate in the whole set of mapping clauses. Each mapping clause is randomly generated by picking a variable in each of the two peers and by negating it with probabilty 0.5 (if the clause is of length 3, the third variable is chosen at random between the variables of the two peers).

The experiments have been conducted on two different platforms. For the measurements concerning the behavior of query processing (number of messages, depth and width of each query processing) we have used a single computer running 1000 peers. Such measurements, consisting in building reports on all peer traces are easier to perform when all data are available on a single computer. In contrast, when time was part of the measurement, we deployed our algorithm on a cluster of 75 heterogenous computers (Athlons with 1GB of RAM: 26 at 1.4GHZ, 9 at 1.8Ghz and 40 at 2 GHZ). In these last experiments, each computer was running around 14 peers, randomly selected.

As we will see, results often exhibit an *exponential* distribution: some queries may need a very long time to complete. It was thus not always possible to perform our experimental analysis without introducing a *timeout* parameter. Each query is labeled with its remaining time to live, which is decreased each time a query needs to traverse a peer to be processed. When necessary, the *timeout* has been set to 30 seconds.

We first report on the distributed behavior of query processing by measuring the number of peers that are involved in query processing. Then, we report on the time of query processing and on the number of answers.

4.1 Distributed behavior of query processing

We have measured the distribution of the depth of query processing as well as the potential width of a query. The *depth* of a query is the maximum length of the reasoning branches developed by the distributed algorithm for returning an answer. The width of the query estimates the number of neighbors that are solicited, on average, for processing a query.

Figure 2 shows the cumulative distribution function of the depth of 1000 queries, when we make vary the number q of shared variables per edge and the ratio $\%3cnf$ of mapping clauses of length 3. Each point on the figure reports a run, for a distinct query.

The four top curves show a relatively small query depth. For instance, with $q = 2$ and $\%3cnf = 0$ no query depth is greater than 7, and none of those four curves have a query depth greater than 36, which suggests that our algorithm behaves well on such networks.

As soon as the value of $\%3cnf$ increases queries have longer depths: with $q = 3$ and $\%3cnf = 20$, 22% of the queries have a depth greater than 100 (the maximum being 134). If we focus on the last three curves on the right, a sharp threshold clearly appears, showing three phases: a sharp growth, representing query processing with small depth, followed by a plateau, and then a slower growth. The small depth query processing and the 'plateau' are characteristics of an exponential distribution of values: most of the processing is easy, but the little remaining is very hard. The slow growth observed is due to the timeout, a side-effect of which is to bound the query depth. Without such a timeout, previous experiments suggest that there would exist some queries requiring very long reasoning branches. This point is outlined on the curve corresponding to the hardest cases ($q = 3$ and $\%3cnf = 20$) where there is no query of depth between 20 and 60. This suggests that when hard processing appears, it is *very hard*. In experiments that are not reported here, we have seen that such an exponential distribution of values was not observed when the acquaintance graphs have a regular ring structure, but was observed on random graphs. We have also measured the *integration degree* of queries, which is the number of *distinct* peers involved in the query processing. We have observed the same kind of exponential distribution of values than for the depth, but with 20% smaller values: 1/5 of the peers are repeated in the histories. That phenomenon was not observed on random acquaintance graphs and seems closely related to the small-world topology.

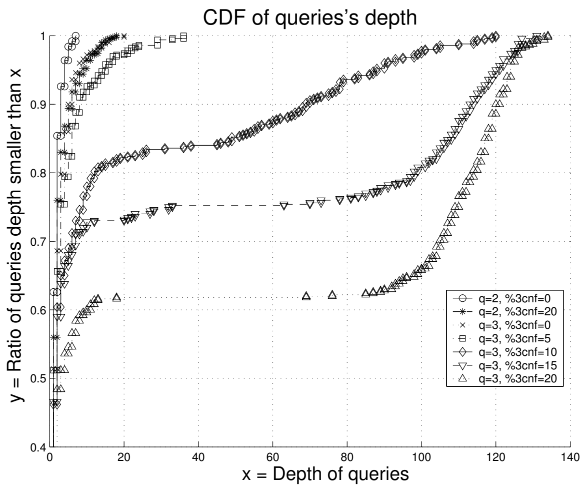

Figure 2: Cumulative Distribution Function of queries depth. The Y scale is re-centered on $[0.4 - 1]$.

Figure 3 shows the cumulative distribution function for the *width* of the queries. Each curve summarizes 20000 runs. With $q = 2$ and $\%3cnf = 0$, more than 75% of the queries are solved locally and 15% other are solved by asking just one neighbor. With $q = 5$ and $\%3cnf = 100$, 25% of the queries solicit 10 neighbors on average, each of them soliciting 10 peers, with 25% chance and so on. Such result explains the combinatorial explosion observed on hard instances.

Our experiments have pointed out a direct impact of the $\%3cnf$ value on query processing, which is not surprising considering the hardness of clauses of length 3 for prime implicate computation. Those experiments also suggest an exponential distribution of query depths, due to the short path length between two peers in the acquaintance graphs, and with an important repetition of solicited peers, due to the large clustering coefficient of small world acquaintance graphs.

4.2 Time and number of answers

We now report a time performance study of our algorithm when it is deployed on a real cluster of 75 heterogeneous

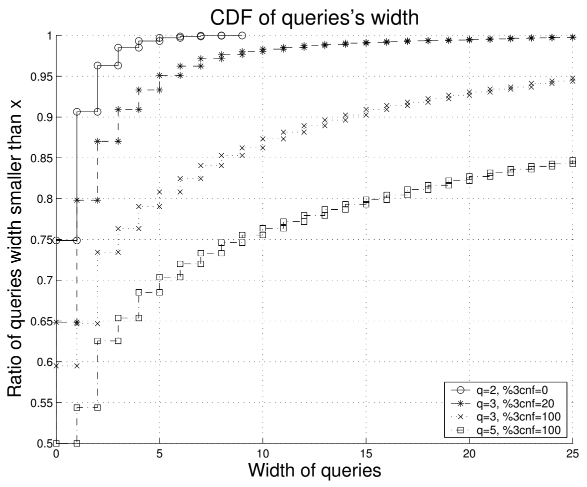

Figure 3: Cumulative Distribution Function of queries width The Y scale is re-centered on $[0.5-1]$, the X axis on $[0-25]$.

computers. Based on the previous observations, we have chosen to focus on 4 differents kinds of acquaintance graphs, denoted *Easy*, *Medium*, *Hard* and *Very Hard* (see Table 1).

	Network			
	Easy $q=2$ $\%3cnf=0$	Medium $q=3$ $\%3cnf=20$	Hard $q=3$ $\%3cnf=100$	Very Hard $q=5$ $\%3cnf=100$
$1^{st} ans.$	0.04s (100%)	1.26s (99.6%)	1.58s (95.6%)	1.39s (89.3%)
$10^{th} ans.$	0.06s (14.3%)	1.37s (25.6%)	0.99s (33.3%)	1.13s (12.0%)
$100^{th} ans.$	–	2.11s (12.7%)	0.84s (27.0%)	4.09s (10.7%)
$1000^{th} ans.$	–	4.17s (6.80%)	4.59s (21.2%)	11.35s (7.15%)
all	0.07s	5.56s	14.6s	21.23s
% timeout	0	13.96	37.5	66.9
#answers	5.17	364	1006	1004
%unsat	4.4	3.64	3.76	1.84

Table 1: Characteristics of the query processing on easy to very hard networks.

The values reported in the Table 1 are mean values over more than 300 different queries. For each column, we show the time needed to produce the 1^{st}, 10^{th}, 100^{th} and 1000^{th} answers of a query. Each mean time is followed by the percentage of initial queries that are taken into account in the averaging. For instance, for the Medium case, 12.7% of the queries have produced more than 100 answers, and the 100^{th} answer was given on average after 2.11 seconds (the average does not take into account queries that did not produce at least 100 answers). The *all* row is the mean time needed to produce all answers, including queries that lead to timeout, the percentage of which is reported in the *%timeout* row. The last two rows report the mean number of answers and the ratio of proven unsatisfiable queries w.r.t. the network.

It is not surprising to find that there is no timeout for the *Easy* case. It is known [Marquis, 2000] that satisfiability checking and prime implicates computation are tractable for sets of clauses of length 2. Moreover, the high partitioning of the global theory induced by the low value of q (number of shared variables) is often a witness of "easy" cases for reasoning for centralized theories. The point to outline for this case is that there are on average 5 answers, and they are produced very quickly.

Even on hard and very hard instances, our algorithm produces a lot of answers coming from a number of different peers. For instance, we measured on average 1006 answers for *Hard*, and 1004 answers for *Very Hard* problems, which already represents a large amount of data. In addition, on those *Very Hard* instances, 90% of runs produced at least one answer. It is noticeable that such hard instance may also be hard for checking the satisfiability of an equivalent, centralized, theory. The formula corresponding to the centralized version of all the distributed theories has n=70 000 variables and m=120 000 clauses 50 000 of which are of length 3. It has been shown in [Monasson *et al.*, 1999] that, for such 2+p formulas, if one does not restrict the locality of variables, the SAT/UNSAT transition is continuous for $p < p_0$, where $p_0 = 0.41$, and discontinuous for $p > p_0$, like in 3-SAT instances. Intuitively, for $p > p_0$, the random 2+p-SAT problem shares the characteristics of the random 3-SAT problems. Let us emphasize here that, with the characteristics of our *Very Hard* network, we have $p = 0.416$ for which the transition phase between SAT and UNSAT instance [Monasson *et al.*, 1999] occurs at m/n=1.69. Here, we have m/n=1.71, which is near the transition phase to confirm that this is where the hard instances would be in practice. Of course, such a comparison is only indicative, because there is no variable locality restriction in the standard 2+p model.

To summarize, when deployed on a real cluster of heterogeneous computers, our algorithm scales very well. Even on very hard instances that shares characteristics of a very large 2+p formula at the crossover between the 2-SAT/3-SAT and the SAT/UNSAT transitions, our algorithm was able to generate a large number of answers in a reasonable time, for a majority of runs.

5 Related work

The distributed message passing algorithm that we have described in Section 3 can be viewed as a distributed version of an Ordered Linear deduction [Chang and Lee, 1973] to produce new target clauses, which was extended by [Siegel, 1987] in order to produce all implicates of a given clause belonging to some target language, and further extended to the first order case in [Inoue, 1992]. The problem of computing new derived clauses (a.k.a. $\langle L, \Phi \rangle$-prime implicates) corresponds exactly to the problem of computing proper prime implicates w.r.t. a theory. It has been extensively studied in the centralized case (see [Marquis, 2000] for a survey).

We have already pointed out the differences between our work and [Amir and McIlraith, 2000]. First, in a full peer-to-peer setting, tree decomposition of the acquaintance graph is not possible. Second, in contrast with partition-based algorithms as in [Amir and McIlraith, 2000], our algorithm is able to *combine* answers from different peers in order to compute implicates involving target variables of different peers.

The model-based diagnosis algorithm for distributed embedded systems [Provan, 2002] exploits the knowledge on the distribution of the system to diagnose for optimization purpose. In distributed ATMS [Mason and Johnson, 1989], agents exchange nogood sets in order to converge to a globally consistent set of justifications. Such a distributed vision of ATMS relies on a global knowledge shared by all the agents and aims at getting a unique global solution. We think that our peer-to-peer inference system can be applied for handling fully distributed model-based diagnosis and fully distributed ATMS, in which no global knowledge is required.

Some recent work deals with distributed first order logic. A model-based and a proof-theoretic semantics has been defined in [Ghidini and Serafini, 1998] for a collection of first order theories communicating through bridge rules that define semantic mappings between their respective domains of interpretation. Based on that work, distributed description logics has been introduced in [Borgida and Serafini, 2003] and a distributed tableau method has been proposed for satisfiability checking.

6 Conclusion

We have presented a peer-to-peer inference system based on propositional logic and we have shown that it scales up to a thousand of peers. The peer-to-peer infrastructure that we have developed is used in a joint project with France Télécom, aiming at enriching web applications with semantics and reasoning services.

So far, we have restricted our algorithm to deal with a vocabulary-based target language. However, it can be adapted to more sophisticated target languages (implicates of a given language, e.g., based on literals and not only variables, of bounded size,...). This can be done by adding a simple tag over all messages to encode the desired target language.

We plan to extend our current work in two main directions. First, we want to tackle Semantic Web applications, in which the bottleneck is to deal with distributed resources shared at large scale. RDF(S) (see [Antoniou and van Harmelen, 2004]) is a standard for annotating web resources, which we think can be encoded in our propositonal setting. Second, we want to handle more sophisticated reasoning in order to deal with a real multi-agent setting, in which possible inconsistencies between agents must be handled.

References

[Adjiman *et al.*, 2004a] P. Adjiman, P. Chatalic, F. Goasdoué, M.-C. Rousset, and L. Simon. Distributed reasoning in a peer-to-peer setting. Technical report, http://www.lri.fr/~goasdoue/bib/ACGRS-TR-1385.pdf, 2004.

[Adjiman *et al.*, 2004b] P. Adjiman, P. Chatalic, F. Goasdoué, M.-C. Rousset, and L. Simon. Distributed reasoning in a peer-to-peer setting, short paper. In *ECAI*, 2004.

[Amir and McIlraith, 2000] E. Amir and S. McIlraith. Partition-based logical reasoning. In *KR*, 2000.

[Antoniou and van Harmelen, 2004] G. Antoniou and F. van Harmelen. *A Semantic Web Primer*. The MIT Press, 2004.

[Borgida and Serafini, 2003] A. Borgida and L. Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics*, 1, 2003.

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

[Dechter and Rish, 1994] R. Dechter and I. Rish. Directed resolution: the davis-putnam procedure revisited. In *KR*, 1994.

[del Val, 1999] A. del Val. A new method for consequence finding and compilation in restricted languages. In *AAAI*, 1999.

[Ghidini and Serafini, 1998] C. Ghidini and L. Serafini. *Distributed First Order Logics*. Research studies Press, 1998.

[Halevy *et al.*, 2003a] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *ICDE*, 2003.

[Halevy *et al.*, 2003b] A. Halevy, Z. Ives, I. Tatarinov, and Peter Mork. Piazza: data management infrastructure for semantic web applications. In *WWW*, 2003.

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

[Marquis, 2000] P. Marquis. *Handbook on Defeasible Reasoning and Uncertainty Management Systems, Algorithms for Defeasible and Uncertain Reasoning*, volume 5, chapter Consequence Finding Algorithms, pages 41–145. Kluwer Academic Publisher, 2000.

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

[Monasson *et al.*, 1999] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. 2+p-sat: Relation of typical-case complexity to the nature of the phase transition. *Random Structure and Algorithms*, 15(414), 1999.

[Nedjl *et al.*, 2002] 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*, 2002.

[Newman, 2000] M. E. J. Newman. Models of the small world. *J. Stat. Phys.*, 101:819–841, 2000.

[Provan, 2002] G. Provan. A model-based diagnosis framework for distributed embedded systems. In *KR*, 2002.

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

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

[Tatarinov and Halevy, 2004] I. Tatarinov and A. Halevy. Efficient query reformulation in peer data management systems. In *SIGMOD 04*, 2004.

[Watts and Strogatz, 1998] D. J. Watts and S. H. Strogatz. Models of the small world. *Nature*, 393:440–442, June 1998.