SomeWhere in the Semantic Web

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

Université Paris Sud XI & CNRS (LRI), INRIA (UR Futurs)**,
Bâtiment 490, Université Paris Sud XI, 91405 Orsay cedex, France

Abstract. In this paper, we describe the SomeWhere semantic peer-to-peer data management system and we provide an experimental analysis showing its scalability. The scalability results from the underlying distributed data model of SomeWhere that can be captured by a fragment of OWL DL, which we think is the appropriate common semantic support needed for most of the future Semantic Web applications (e.g., those based on taxonomies). In this setting, query answering over peers can be done by distributed query rewriting, which can be equivalently reduced to distributed reasoning over logical propositional theories.

1 Introduction

The Semantic Web [1] envisions a world-wide distributed architecture where data and computational resources will easily inter-operate to coordinate complex tasks such as answering queries or global computing. Semantic marking up of web resources using *ontologies* is expected to provide the necessary glue for making this vision work. The de-centralized nature of the Web makes inevitable that communities of users or software developers will use their own ontologies to describe their data or services. This vision of the Semantic Web corresponds to semantic peer-to-peer networks, in which each peer can indistinctly play the role of data (or service) provider and of mediator with other peers. The problem of schema mediation in peer-to-peer data management systems has been investigated recently in [2], where mappings between relational schemas are expressed using a powerful relational formalism. It is shown that in this setting, query answering is undecidable except if some restrictions are imposed on the mappings and on the resulting topology of the peer-to-peer network.

In this paper, we describe the SomeWhere semantic peer-to-peer data management system and we provide an experimental analysis showing its scalability. The scalability results from the underlying distributed data model of SomeWhere which can be captured by the fragment of OWL DL [3] corresponding to logical combination of atomic classes using the conjunction, disjunction, and complement constructors. In particular, this model allows for expressing taxonomies and mappings between taxonomies, which we think are the appropriate common semantic support needed for most of the future semantic web applications. In this setting, query answering over multiple schemas can be done by distributed query rewriting, which can be equivalently reduced to distributed reasoning over logical propositional theories.

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

The paper is organized as follows. Section 2 defines the SomeWhere data model and the corresponding query answering and rewriting problems. In Section 3, we show the propositional encoding allowing us to equivalently reduce the distributed computation of query rewritings to a distributed reasoning task in propositional logic. In Section 4, we describe the message based distributed reasoning algorithm that is implemented in SomeWhere. Section 5 reports experimental results for evaluating the scalability of SomeWhere. We conclude with related work and a short discussion in Section 6.

2 Data model and query answering in SomeWhere

SomeWhere is a network of peers having their own terminology of classes for describing their schema and stored data, but being semantically related by mappings. A new peer joins the peer-to-peer network through some peers that it knows (called its acquaintances) by declaring mappings between its own terminology and the terminologies of its acquaintances. Queries are posed to a given peer using its local terminology. The answers that are expected are not only instances of local classes but possibly instances of classes of peers distant from the queried peer if it can be infered from the peer ontologies and the mappings that they satisfy the query. Local terminologies, data descriptions and mappings are defined using a fragment of OWL DL that we will denote OWL PL, where PL stands for propositional logic. We first present OWL PL, then we introduce the peer-to-peer SomeWhere data model and query answering problem.

2.1 Syntax and semantics of OWL PL

OWL PL is the fragment of OWL DL reduced to the union, intersection and complement constructors for building class descriptions. An ontology is made of a set of axioms and facts. The axioms that are allowed in OWL PL are class (partial or complete) definitions that associate class identifiers with class descriptions, and disjointness, equivalence or inclusion statements between class descriptions. The facts that are allowed in OWL PL are restricted to statements relating individual identifiers to class identifiers. The OWL PL syntax is defined as follows as an OWL DL sub-grammar:

```
ontology ::= 'Ontology(' [ontologyID] {directive} ')'
directive ::= axiom | fact
axiom ::= 'Class(' classID modality description {description} ')'
        | 'DisjointClasses(' description description {description} ')'
        | 'EquivalentClasses(' description description {description} ')'
        | 'SubClassOf(' description description ')'
modality ::= 'complete' | 'partial'
description ::= classID
        | owl:Thing
        | 'unionOf(' description description {description} ')'
        | 'intersectionOf(' description description {description} ')'
        | 'complementOf(' description ')'
fact ::= 'Individual(' individualID {'type(' classID ')'} ')'
```

The semantics of OWL PL is a standard logical formal semantics defined in terms of *interpretations*. An interpretation I is a pair $(\Delta^I, .^I)$ where Δ is a non-empty set, called the domain of interpretation, and $.^I$ is an interpretation function which assigns a subset of Δ^I to every class identifier and an element of Δ^I to every individual identifier.

Definition 1 (Model of an ontology (or a collection of ontologies)). *An interpretation I is a* model *of an ontology O (respectively of a collection of ontologies $\{O_i\}_{i=1}^n$) iff each directive in O (respectively in $\{O_i\}_{i=1}^n$) is satisfied by I.*

Directives are satisfied in OWL PL if the following holds:
- if $Class(classID\ complete\ d_1 \dots d_n)$ is in O then $classID^I = \bigcap_{i=1}^n (d_i)^I$
- if $Class(classID\ partial\ d_1 \dots d_n)$ is in O then $classID^I \subseteq \bigcap_{i=1}^n (d_i)^I$
- if $DisjointClasses(d_1 \dots d_n)$ is in O then $(d_i)^I \cap (d_j)^I = \emptyset$ for every $i, j\ 1 \leq i < j \leq n$
- if $EquivalentClasses(d_1 \dots d_n)$ is in O then $(d_i)^I = (d_j)^I$ for every $i, j\ 1 \leq i < j \leq n$
- if $SubClassOf(d_1\ d_2)$ is in O then $(d_1)^I \subseteq (d_2)^I$
- if $Individual(individualID\ type(t1) \dots type(t_n))$ is in O then $individualID^I \in \bigcap_{i=1}^n (t_i)^I$

Interpretations of axioms rely on interpretations of descriptions which are inductively defined as follows:
- $(owl : Thing)^I = \Delta^I$
- $(unionOf(d_1 \dots d_n))^I = \bigcup_{i=1}^n (d_i)^I$
- $(intersectionOf(d_1 \dots d_n))^I = \bigcap_{i=1}^n (d_i)^I$
- $(complementOf(d_1))^I = \Delta^I \backslash (d_1)^I$

Definition 2 (Satisfiability and entailment). *An ontology (or a collection of ontologies) is* satisfiable *iff it has a model. An ontology (or a collection of ontologies)* entails *an OWL directive if every model of the ontology (or of the collection of ontologies) is a model of the directive.*

Definition 3 (Class subsumption (wrt ontologies)). *Let C and C' be class descriptions.*
- *C* is subsumed *by C' if for every interpretation I C^I is a subset of C'^I.*
- *Let O (respectively $\{O_i\}_{i=1}^n$) be an ontology (respectively a collection of ontologies). C* is subsumed *by C'* wrt *O (respectively* wrt *$\{O_i\}_{i=1}^n$) if for every model I of O (respectively of $\{O_i\}_{i=1}^n$) C^I is a subset of C'^I.*

2.2 SomeWhere peer-to-peer data model

In a SomeWhere peer-to-peer network, the content of each peer is defined by an OWL PL ontology from which we exhibit a terminological component, a view component, an assertional component and a mapping component. The *terminological component* is made of class definitions and possibly disjunction, inclusion or equivalence statements. The class identifiers appearing in the terminology will be referred to as *intentional classes*. The *view component* is made of the class definitions for which individuals are given in the assertional component. Class identifiers of these definitions will be referred to as *extensional classes*. The views component is the storage description of the peer. We impose some constraints on extensional classes to fit with a $Local\_as\_Views$ approach and an open-world assumption of the information integration setting: extensional classes must be partially defined in terms of intentional classes only and they cannot appear in any components in descriptions involved in disjunction, inclusion or equivalence axioms. The *assertional component* is made of the fact statements that relate individuals to extentional classes of the view component. The *mapping component* is made of the set of disjunction, inclusion or equivalence statements relating intentional

classes of a peer P (called *local classes* wrt P) with intentional classes belonging to some other peers (called *foreign classes* wrt P). Those mappings are in the ontology of P because when P joint the peer-to-peer network, it had to know some peers already in the network and it had to declare some semantic connections between its terminological component and those of its acquaintances.

In the following, we will denote the ontology of a peer P by $\langle \mathcal{T}, \mathcal{V}, \mathcal{A}, \mathcal{M} \rangle$ where $\mathcal{T}$, $\mathcal{V}$, $\mathcal{A}$ and $\mathcal{M}$ are respectively the terminological, view, assertional and mapping components of P. We call the *vocabulary* of a peer the set of (intentional or extensional) class identifiers appearing in its terminological, view and mapping components. We will denote the content of a SomeWhere peer-to-peer network $\mathcal{P}$ made of a collection of peers $\{P_i\}_{i=1}^n$ by the collection of ontologies $\{\langle \mathcal{T}_i, \mathcal{V}_i, \mathcal{A}_i, \mathcal{M}_i \rangle\}_{i=1}^n$, where each $\langle \mathcal{T}_i, \mathcal{V}_i, \mathcal{A}_i, \mathcal{M}_i \rangle$ is the ontology of peer P_i. Without loss of generality we assume that the class identifiers defined by the different peers of a network are distinct.

In a SomeWhere network, there is no centralized schema but a distributed schema made of the terminological, view and mapping components of the different peers.

Definition 4 (Data and Schema of a SomeWhere peer-to-peer network).
Let $\mathcal{P}$:$\{\langle \mathcal{T}_i, \mathcal{V}_i, \mathcal{A}_i, \mathcal{M}_i \rangle\}_{i=1}^n$ be a SomeWhere peer-to-peer network.
- *The* schema $S(\mathcal{P})$ *of $\mathcal{P}$ is the collection of ontologies $\{\langle \mathcal{T}_i, \mathcal{V}_i, \emptyset, \mathcal{M}_i \rangle\}_{i=1}^n$.*
- *The* data $D(\mathcal{P})$ *of $\mathcal{P}$ is $\{a \mid Individual(a\ type(t1) \ldots type(t_n)) \in \bigcup_{i=1}^n \mathcal{A}_i\}$.*

The point is that the data and the schema of a peer-to-peer network are distributed and each peer has a partial knowledge of it: it knows its own data and schema, and mappings with its acquaintances. The *acquaintance graph* accounts for the connection induced by the mappings between the different peers within a given SomeWhere peer-to-peer network.

Definition 5 (Acquaintance graph). *Let $\mathcal{P} = \{P_i\}_{i=1}^n$ a collection of peers with their respective vocabularies Voc_{P_i}. Let $Voc = \bigcup_{i=1}^n Voc_{P_i}$ be the vocabulary of $\mathcal{P}$. Its* acquaintance graph *is a graph $\Gamma = (\mathcal{P}, \text{ACQ})$ where $\mathcal{P}$ is the set of vertices and $\text{ACQ} \subseteq Voc \times \mathcal{P} \times \mathcal{P}$ is a set of labelled edges such that for every $(c, P_i, P_j) \in \text{ACQ}$, $i \neq j$ and $c \in Voc_{P_i} \cap Voc_{P_j}$.*

A labelled edge (c, P_i, P_j) expresses that peers P_i and P_j know each other to be sharing the class c. This means that c belongs to the intentional classes of P_i (or P_j) and is involved in a mapping with intentional classes of P_j (or P_i).

Example Let us consider four persons Ann, Bob, Chris and Dora, each having some data about restaurants they know or like. Each of them maintains his/her own terminology and views. Ann, who is working as a restaurant critics, distinguishes restaurants considered as offering a "good" cooking (G), among which are those which are rated (R) with $1, 2$ or 3 stars ($S1, S2, S3$). Ann also records the kind of cooking. She distinguishes Indian cooking (I) from oriental cooking (O), which for her covers Chinese (C), Taï (T) and Vietnamese (V) cooking. Bob is found of Asian cooking (A) and also keeps track of what he considers to be the best quality (Q) restaurants he knows. Chris is more found of fish restaurants (F) but recently discovered some places serving a very nice cantonese cuisine (CA). Eventually, Dora's preferred restaurants (DP) are

pizzeria (P), as well as restaurants serving Asian cuisine (but only of good quality) or seafood (SF).

Ann, Bob, Chris and Dora are modelled as four peers P_1, P_2, P_3 and P_4 As a convention, we use the previously introduced symbols to denote intentional classes and symbols of the form MyX to denote extensional classes. We suppose that Ann stores data on restaurants of various specialties, but that the only rated restaurants for which she has stored data are those rated with 2 stars. We consider that Dora only keeps data on pizzerias. We do not describe explicitly the assertional part of the peers but assume that to each view defined in $\mathcal{V}_i$ corresponds at least one stored individual.

Peer 1 :

$\mathcal{T}_1$
```
Class(G partial Thing)
Class(S1 partial Thing)
Class(S2 partial Thing)
Class(S3 partial Thing)
DisjointClasses(S1 S2 S3)
Class(R complete unionOf(S1 S2 S3))
SubClassOf(R G)
Class(O partial Thing)
Class(I partial Thing)
Class(C partial O)
Class(V partial O)
Class(T partial O)
```

$\mathcal{V}_1$
```
Class(MyS2 partial S2)
Class(MyC partial C)
Class(MyV partial V)
Class(MyT partial T)
Class(MyI partial I)
```

Peer 2 :

$\mathcal{T}_2$
```
Class(A partial Thing)
Class(Q partial Thing)
```

$\mathcal{V}_2$
```
Class(MyA partial A)
Class(MyQ partial Q)
```

Peer 3 :

$\mathcal{T}_3$
```
Class(F partial Thing)
Class(CA partial Thing)
```

$\mathcal{V}_3$
```
Class(MyF partial F)
Class(MyCA partial CA)
```

Peer 4 :

$\mathcal{T}_4$
```
Class(DP partial Thing)
Class(P partial DP)
Class(SF partial DP)
```

$\mathcal{V}_4$
```
Class(MyP partial P)
```

The mappings express what each peer knows about its acquaintances:

Peer 1 Ann knows that Cantonese cuisine is a particular case of Chinese cooking. Moreover she is very confident in Bob's taste and agree to include Bob' selection as good restaurants. She does'nt know exactly what Asian means for Bob, but think that this notion encompasses her concept of oriental restaurant.

$\mathcal{M}_1$: `SubClassOf(CA C)` `SubClassOf(Q G)` `SubClassOf(O A)`

Peer 2 Bob knows that what he calls Asian cooking exactly corresponds to what Ann classifies as Oriental cooking

$\mathcal{M}_2$: `EquivalentClasses(A O)`

Peer 3 Chris considers that fish specialties are a particular case of seafood specialties.

$\mathcal{M}_3$: `SubClassOf(F SF)`

Peer 4 Dora counts on Ann and Bob's knowledge to obtain good Asian restaurants.

$\mathcal{M}_4$: `SubClassOf(intersectionOf(A G) DP)`

Figure 1 describes the acquaintance graph corresponding to this example. We just mention the vocabulary of each peer. Edges are labeled with the class identifiers that are shared (through mappings) between peers.

2.3 Query answering in SomeWhere

In SomeWhere, each user interrogates the peer-to-peer network through one peer of his choice, and uses the vocabulary of this peer to express his query.

Definition 6 (Query). *Let $\mathcal{P}$ be a SomeWhere peer-to-peer network. A* query *is an OWL PL description in terms of classes of a particular peer of $\mathcal{P}$.*

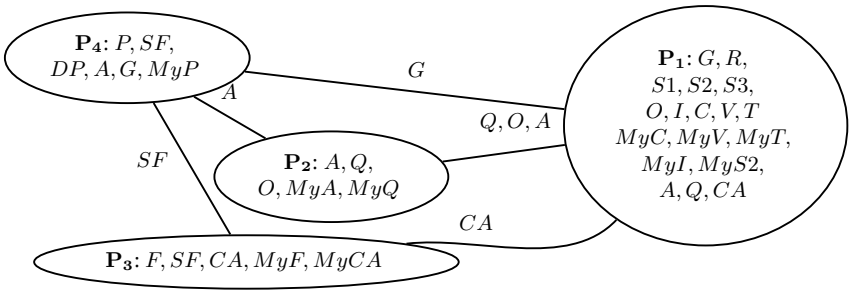

Fig. 1. The acquaintance graph

The point is that, even if the query is local to a peer, the expected answers are all the individuals, possibly stored at different peers, which satisfy the query. The following definition provides the formal semantics of query answering in SomeWhere: an individual a is an answer for the query q if $q(a)$ is entailed by the union of the OWL PL ontologies of all the peers in the network.

Definition 7 (answers). *Let $\mathcal{P}$:$\{\langle \mathcal{T}_i, \mathcal{V}_i, \mathcal{A}_i, \mathcal{M}_i \rangle\}_{i=1}^n$ be a SomeWhere peer-to-peer network. Let q be a query defined over a peer of $\mathcal{P}$. Let a be an individual of $D(\mathcal{P})$: a is an* answer *of q iff $a^I \in q^I$ for every model I of $\{\langle \mathcal{T}_i, \mathcal{V}_i, \mathcal{A}_i, \mathcal{M}_i \rangle\}_{i=1}^n$.*

Given a SomeWhere peer-to-peer network $\mathcal{P}$ and a query q, the query answering problem is to find all the answers of q. In general, finding all answers in a peer data management system is a critical issue [2]. In our setting however, we are in a case where all the answers can be obtained using *rewritings* of the query. We first define the notion of (maximal) conjunctive rewriting of a query in our setting. We then show how conjunctive rewritings can be used to compute all the answers for the query.

Definition 8 (Conjunctive rewriting of a query). *Let $\mathcal{P}$ be a SomeWhere peer-to-peer network and $S(\mathcal{P})$ its schema. Let q be a query asked to a peer of $\mathcal{P}$. Let q_e be a query defined as an intersection of extensional classes only. q_e is a* conjunctive rewriting *of q iff q_e is subsumed by q wrt $S(\mathcal{P})$. It is* maximal *iff there does not exist a strict subsumer of q_e which is a rewriting of q.*

It results from this definition that the answers of conjunctive rewritings of a query q are answers for q. Getting the answers of a conjunctive rewriting is straightforward: the answer set of q_e is obtained as the intersection of the extensions of the extensional classes in its definition. Thanks to the restrictions that we have imposed to the extensional classes, the extension of an extensional class $classID$ is simply the set of the individual identifiers declared as of type $classID$ in the corresponding ontology. Most importantly, it has been shown in [4, 5] that when a query has a *finite number of maximal conjunctive rewritings*, then *all* its answers can be obtained in polynomial data complexity as the union of the answer sets of its rewritings.

In the following section, we present a *propositional encoding* that enables us to show that in our setting every query has a finite number of maximal conjunctive rewritings, and that maximal conjunctive rewritings correspond exactly to the negation of proper prime implicates of the negation of the query in the distributed logical theory resulting from that encoding.

3 Propositional encoding of query rewriting in SomeWhere

The propositional encoding concerns the schema of a SomeWhere peer-to-peer network and the queries. It consists in transforming each query and schema statement into a propositional formula using class identifiers as propositional variables.

The propositional encoding of an OWL PL description d, and thus of a query, is the propositional formula $Prop(d)$ obtained inductively as follows:

- $Prop(owl:Thing) = true$
- $Prop(A) = A$, if A is a class identifier
- $Prop(unionOf(d_1 \dots d_n)) = \bigvee_{i=1}^{n}(Prop(d_n))$
- $Prop(intersectionOf(d_1 \dots d_n)) = \bigwedge_{i=1}^{n}(Prop(d_n))$
- $Prop(complementOf(d_1)) = \neg(Prop(d_1))$

The propositional encoding of the schema $S(\mathcal{P})$ of a SomeWhere peer-to-peer network $\mathcal{P}$ is the distributed propositional theory $Prop(S(\mathcal{P}))$ made of the formulas obtained inductively from the OWL PL directives of $S(\mathcal{P})$ as follows:

- $Prop(Class(classID\ complete\ d_1 \dots\ d_n)) = classID \Leftrightarrow \bigwedge_{i=1}^{n}(Prop(d_i))$
- $Prop(Class(classID\ partial\ d_1 \dots\ d_n)) = classID \Rightarrow \bigwedge_{i=1}^{n}(Prop(d_i))$
- $Prop(DisjointClasses(d_1 \dots d_n)) = \bigwedge_{i=1}^{n}\bigwedge_{i<j}^{n}(\neg(Prop(d_i)) \vee \neg(Prop(d_j)))$
- $Prop(EquivalentClasses(d_1 \dots d_n)) = \bigwedge_{i=1}^{n}\bigwedge_{i<j}^{n}(Prop(d_i) \Leftrightarrow Prop(d_j))$
- $Prop(SubClassesOf(d_1\ d_2)) = Prop(d_1) \Rightarrow Prop(d_2))$

From now on, for simplicity purpose, we use the propositional clausal form notation for the queries and SomeWhere peer-to-peer network schemas.

As an illustration, let us consider the propositional encoding of the example presented in Section 2.2. After application of the transformation rules, conversion of each produced formula in clausal form and suppression of tautologies, we obtain (figure 2) a new acquaintance graph where each peer schema is described as a propositional theory.

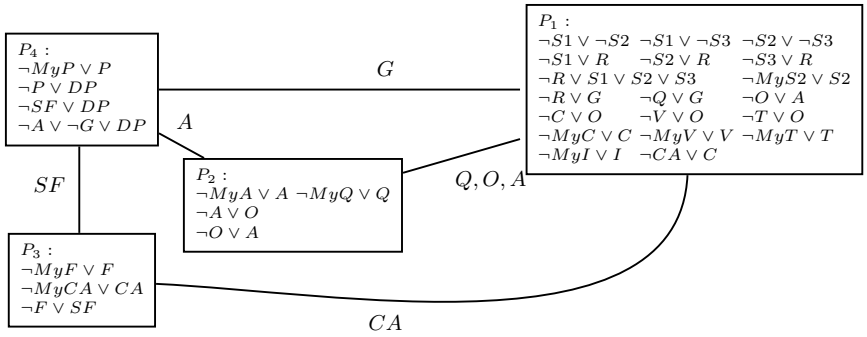

Fig. 2. The propositional encoding for the restaurant network

Proposition 1 states that the propositional encoding transfers satisfiability and establishes the connection between (maximal) conjunctive rewritings and clausal proper (prime) implicates.

Definition 9 (Proper prime implicate wrt a theory). *Let T be a clausal theory and q be a clause. A clause m is said to be:*

- *a* prime implicate *of q wrt T iff $T \cup \{q\} \models m$ and for any other clause m', if $T \cup \{q\} \models m'$ and $m' \models m$ then $m' \equiv m$.*
- *a* proper prime implicate *of q wrt T iff it is a prime implicate of q wrt T but $T \not\models m$.*

Proposition 1 (Propositional transfer). *Let $\mathcal{P}$ be a SomeWhere peer-to-peer network and let $Prop(S(\mathcal{P}))$ be the propositional encoding of its schema. Let V_e be the set of all the extensional classes.*

- *$S(\mathcal{P})$ is satisfiable iff $Prop(S(\mathcal{P}))$ is satisfiable.*
- *q_e is a maximal conjunctive rewriting of a query q iff $\neg Prop(q_e)$ is a* proper prime implicate of $\neg Prop(q)$ wrt $Prop(S(\mathcal{P}))$ *such that all its variables are extensional classes.*

As a result we can use any SAT algorithm for checking satisfiability of a SomeWhere peer-to-peer network schema. Most importantly, Proposition 1 gives us a way to compute *all* the answers of a query. The maximal conjunctive rewritings of a query q within a peer-to-peer network $\mathcal{P}$ correspond to negation of the proper prime implicates of $\neg q$ wrt the propositional encoding of the schema of $S(\mathcal{P})$. Since the number of proper prime implicates of a clause wrt a clausal theory is finite, every query in SomeWhere has a finite number of maximal conjunctive rewritings. Therefore, according to [4, 5], the set of *all* of its answers is exactly the union of the answer sets of its rewritings.

In the following section, we exhibit a distributed consequence finding algorithm which computes the set of proper prime implicates of a literal wrt a distributed propositional clausal theory. According to Proposition 1, if this algorithm is applied to a distributed theory resulting from the propositional encoding of the schema of a SomeWhere peer-to-peer network, with the extensional classes considered as target variables, and triggered with a literal $\neg q$, it computes in fact the negation of the maximal conjunctive rewritings of the *atomic* query q. Since in our setting the maximal rewritings of an arbitrary query can be obtained by combining the maximal rewritings of its atomic components, we focus on the computation of the rewritings of atomic queries.

Illustration on the example Before going into the details of the algorithm, we illustrate its behavior on finding the conjunctive rewritings of the query DP asked to P_4 for finding Dora's preferred restaurants. The algorithm computes the proper implicates of $\neg DP$ starting from P_4. The local proper implicates of $\neg DP$ with respect to P_4 that are produced are: $\{\neg MyP, \neg P, \neg SF, \neg A \vee \neg G\}$. The clause $\neg MyP$ only contains one extensional class symbol. Its negation MyP is thus a first *local rewriting*. The clause $\neg P$ is discarded since it does not contain any symbol shared whith some other peer. But the last two clauses can be used to try to find further rewritings, by soliciting P_4 acquaintances that share symbols with these clauses. The clause $\neg SF$ is transmitted for propagation to P_3, which produces $\{\neg MyF, \neg F\}$. Since $\neg MyF$ only contains one extensional class symbol, its negation MyF is a second rewriting. We call it a *remote rewriting* because it has been produced by inference through a remote peer . Since F is not shared, $\neg F$ is discarded and this branch of reasoning is closed.

The last clause to consider is $\neg A \vee \neg G$ which contains two shared symbols. Our algorithm splits such a clause and separately computes implicates of each of its components ($\neg A$ and $\neg G$). The respective results will be recombined afterwards. We first focus on $\neg A$, which is transmitted for propagation to P_2. This produces the clauses $\{\neg MyA, \neg O\}$, among which $\neg MyA$ is left pending for further combination. $\neg O$ and $\neg A$ are transmitted to P_1. The propagation of $\neg O$ produces as new proper implicates of $\neg A$: $\neg MyV$, $\neg MyT$ and $\neg MyC$, as well as $\neg MyCA$ (by further propagation of $\neg C$ in P_3). Transmitting $\neg A$ to P_1 produces again the clause $\neg O$ in P_1 (by resolution with

$\neg O \vee A$). Since O is shared with P_2, $\neg O$ is transmitted back to P_2. Its propagation in P_2 produces $\neg A$. At this stage, the algorithm checks in its history that $\neg A$ has already been transmitted to P_2 before and stops the current reasoning branch. Handling such histories is the means for avoiding the algorithm to loop. The whole set of proper implicates of $\neg A$ is then: $\{\neg MyA, \neg MyV, \neg MyT, \neg MyC, \neg MyCA\}$. In a similar way, transmitting $\neg G$ to P_1 produces as proper imlicates of $\neg G$: $\{\neg MyS2, \neg MyQ\}$. Recombining the implicates of each component of the splitted clause amounts to building all the clauses of the product of $\{\neg MyS2, \neg MyQ\}$ by $\{\neg MyA, \neg MyV, \neg MyT, \neg MyC, \neg MyCA\}$. The negation of each of the 10 clauses produced in this way (e.g. $MyS2 \wedge MyA$, $MyQ \wedge MyT$,..) constitutes new rewritings of the initial query. We call such rewritings *integration rewritings*. Eventually, the inital query admits a total of 12 maximal conjunctive rewritings. For example, the rewriting $MyS2 \wedge MyA$ states that the identifiers of restaurants that are found both in the data stored by Ann as 2 stars rated restaurants and in the data stored by Bob as Asian cooking restaurants are answers to the query asking for Dora's preferred restaurants.

4 Distributed consequence finding algorithm

The distributed algorithm that we have designed is a message passing algorithm implemented locally at each peer. It handles 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 algorithm is composed of three procedures, each one being triggered by the reception of a message.

The procedure RECEIVEQUERYMESSAGE is triggered by the reception of a $query$ message $m(Sender, Receiver, query, hist, l)$ sent by the peer $Sender$ to the peer $Receiver$ which executes the procedure: on the demand of $Sender$, with which it shares the variable of l, it processes the literal l.

The procedure RECEIVEANSWERMESSAGE is triggered by the reception of an $answer$ message $m(Sender, Receiver, answer, hist, r)$ sent by the peer $Sender$ to the peer $Receiver$ which executes the procedure: it processes the answer r (which is a clause the variables of which are target variables) sent back by $Sender$ for the literal l (last added in the history) ; it may have to combine it with other answers for literals being in the same clause as l.

The procedure RECEIVEFINALMESSAGE is triggered by the reception of a $final$ message $m(Sender, Receiver, final, hist, true)$: the peer $Sender$ notifies the peer $Receiver$ that answer computation for the literal l (last added in the history) is completed.

Those procedures handle two data structures stored at each peer: ANSWER$(l, hist)$ caches the answers resulting from the propagation of l within the reasoning branch corresponding to $hist$; FINAL$(q, hist)$ is set to true when the propagation of q within the reasoning branch of the history $hist$ is completed. 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 proce-

dures, since they are locally executed by the peer which receives the message, we will denote by $Self$ the receiver peer.

In the following, we will use the notations:

- for a literal q, $Resolvent(q, P)$ denotes the set of clauses obtained by resolution between q and a clause of P,
- for a literal q, $\bar{q}$ denotes its complementary literal,
- 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. The condition $S(c) = \square$ thus expresses that c does not contain any shared variable,
- $\otimes$ is the distribution operator on sets of clauses: $S_1 \otimes \cdots \otimes S_n = \{c_1 \vee \cdots \vee c_n \mid c_1 \in S_1, \ldots, c_n \in S_n\}$. If $L = \{l_1, \ldots, l_p\}$, we will use the notation $\otimes_{l \in L} S_l$ to denote $S_{l_1} \otimes \cdots \otimes S_{l_p}$.
- $\mathcal{T}arget(P)$ is the language of clauses (including the empty clause) involving only variables that are extensonial classes of P.

Algorithm 1: Message passing procedure for processing queries

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$(Self) \mid L(c) \in \mathcal{T}arget(Self)\}$
(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** $m(Self, RP, query, [(q, Self, c)|hist], l)$

The following theorems summarize the main properties of our distributed message passing algorithm. Theorem 1 states the termination and the soundness of the algorithm. Theorem 2 states its completeness under the condition that each local theory is saturated

Algorithm 2: Message passing procedure for processing answers
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$\leftarrow \otimes_{l \in S(c) \setminus \{l'\}}$ ANSWER$(l, hist) \otimes \{L(c) \vee r\}$
(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: Message passing procedure for notifying termination
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$(l, [(l, Sender, _), (q, Self, c)|hist']) \leftarrow \emptyset$

by resolution. Theorem 3 states that the user is notified of the termination when it occurs, which is crucial for an anytime algorithm. Their full proofs are given in [6], in which the properties of termination, soundness and completeness are first proven for a recursive consequence finding algorithm that we do not provide in this paper by lack of space. Theorem 1 and Theorem 2 are then obtained by proving that the distributed message passing algorithm described in this paper terminates and computes the same results as that recursive algorithm. It is important to note that the recursive consequence finding algorithm described in [6] applies in a general setting of distributed theories sharing variables. As a result, its completeness is only guaranteed if the acquaintance graph satisfies a certain property: if two local theories have a common variable, there must exist in the acquaintance graph a path between those two theories, all the edges of which are labeled with that variable. In our setting, the variables that may be common to two theories necessarily comes from the encoding of mapping statements, and the above property is always satisfied by the acquaintance graphs resulting from the propositional encoding of the schemas of SomeWhere peer-to-peer networks.

In the following theorems, let $\mathcal{T}$ be the propositional encoding of the schema $S(\mathcal{P})$ of a peer-to-peer SomeWhere network, let $\neg q$ the negation of an atomic query q, let T be the propositional encoding of the local schema and mappings of the asked peer.

Theorem 1. *If T receives from the user the message $m(User, T, query, \emptyset, \neg q)$, then:*
- *a finite number of answer messages will be produced ;*
- *each produced answer message $m(T, User, answer, [(\neg q, T, \_)], r)$ is such that r is an implicate of $\neg q$ wrt $S(\mathcal{P})$ which belong to $\mathcal{T}arget(\mathcal{P})$.*

Theorem 2. *If each local theory is saturated by resolution and if T receives from the user the message $m(User, T, query, \emptyset, \neg q)$, then for each proper prime implicates r of $\neg q$ wrt $S(\mathcal{P})$ belonging to $\mathcal{T}arget(\mathcal{P})$, an answer message $m(T, User, answer, [(\neg q, T, \_)], r)$ will be produced.*

Theorem 3. *If r is the last result returned through an answer message $m(T, User, answer, [(\neg q, T, \_)], r)$ then the user will be notified of the termination by a message $m(T, User, final, [(\neg q, T, true)], true)$.*

5 Experimental analysis

Evaluating the performances of SomeWhere peer-to-peer networks amounts to evaluate our message passing algorithm on propositional encodings of OWL PL ontologies. Experimental data corresponding to such encodings are missing, therefore our experiments have been performed on random generated distributed propositional theories. Instances used during experiments may be characterized by the structure of the acquaintance graph and the structure of peer propositional theories. The acquaintance graphs used in all instances are uniform random connected graphs, characterized by the number d of nodes and e edges, $c = e/d$ being the graph's *connection ratio*. For all instances, peer theories are generated randomly using the same model, n denotes the number of variables in each theory, p the number of (randomly chosen) target variables, and q the number of variables shared by connected peers (we fixed $q = 2$ in all instances).

SomeWhere has been developed in Java. All experiments have been done on two clusters of Linux machines with 1Gb memory[1]. Two kinds of experiments have been performed. In the first one, peer theories correspond to uniform random 3-CNF theories, that have been widely studied in the propositional reasoning community and for which the problem of prime implicates computation is known to be hard [7, 8]. The main goal of these experiments is to evaluate the robustness of the SomeWhere architecture in the context of large networks of complex theories. Because of the tremendous amount of possible rewritings in such networks, we only study those obtained in a limited time. However, random 3-CNF theories are not necessarily representative of taxonomy-like encodings. Therefore, a second kind of experiments has been conducted, intended to be *more realistic*, in which peer theories encode random trees.

5.1 3-CNF theories, witnesses for robustness

In the first experiments each peer contains a set of m random clauses of length 3 (on n variables) and additional equivalence clauses relating variables shared by connected peers. Although it has received little attention until now, proper prime implicate computation is a difficult problem. As an illustration, the figure 3.a describes, for $m = 30$ and different values of n, the average values of the total number of proper prime implicates (# PPI) and of their distribution according to their size. Since we wanted to connect a large number of such peers we fixed m to 30 and n to 15 for each peer (i.e., an average of 20 PPI for each peer, about half of which contain more than 3 literals). We asked 1000 random queries equally distributed on all the peers of Cluster1. Because of the inherent intractibility of the global theory, we could not expect all the queries to finish. We tried to run `zres` [8], a state of the art prime implicate solver, on the (centralized) global theory (14392 clauses, 5400 variables) but it blowed-up with billions of clauses. Therefore, we used a time cut off of 90s, which seems to be a good candidate treshold for human interactive queries.

Results are summarized on figure 3.b The plain curve (first one in the legend) represents the (average) total number of rewritings returned within the elapsed time. The two other curves correspond to (normalized) cumulative distribution function (CDF)

[1] Cluster1: 18 diskless Pentium IV 2800 MHz and Cluster2: 71 Athlons($>$1800 MHz)

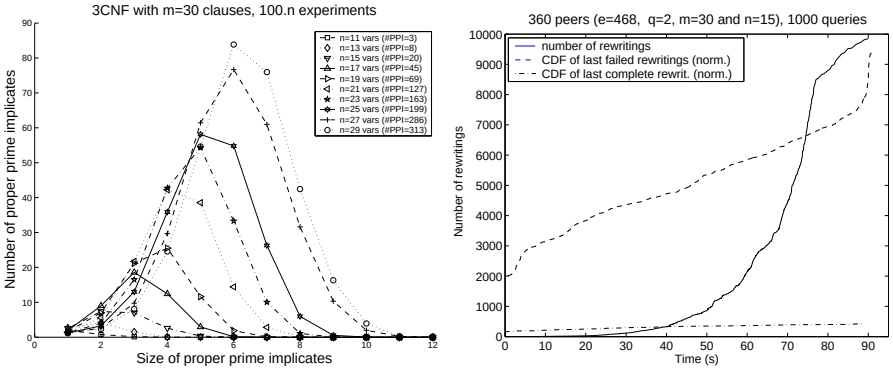

Fig. 3. (a), left figure, Proper Prime Implicates characteristics in a (single) random 3CNF theory; and (b), right figure, speed-up of rewritings on distributed 3-CNF theories

of the last returned rewriting. We distinguish *completed queries* (i.e. those terminated within the time cutoff) from *failed queries* (i.e. still waiting for further results at time cutoff). The failed curve (second one) thus characterizes, for a given time, the proportion of queries that will not obtain any new rewriting within the remaining time. For instance, after 20s 37% of the queries are waiting for results from other peers and will not obtain further answer before the 90s limit. The last curve describes the proportion of queries completed at a given time (before 90s). The first striking result is the exponential variation of the number of rewritings. Waiting a little bit longer may result in an exponentially increasing number of new rewritings. The important part of queries that quickly "fail" (waiting for rewritings from the beginning) may be explained by rewritings of clauses with shared variables but no target variable. They need a lot of time but do not return any result. However, it is interesting to note that this curve is linear. Half of the queries were still producing new results after $50s$.

Another interesting result concerns the origin of target literals in the returned rewritings. We observed that the mean number of peers producing these literals is about 8. In average, the total number of literals composing the returned rewritings is 6157. Among them, 2381 come from a first peer, 1810 from second peer, ... but only 1 comes from an 8th peer. However the median values are much smaller (688 for a first peer, 483 for a second peer,...). This suggests an exponential behavior where some of the produced rewritings are quite large. Moreover we observed that the median number of peers making up the rewritings is more than two times greater than the graph degree, which suggests that more than half of the rewritings were composed by at least two degrees of knowledge (needing friends of friends to produce rewritings).

5.2 Distributed theories encoding trees

In the second experiments, each peer theory corresponds to the CNF encoding of a random tree and contains only binary clauses ($a \vee \neg b$ encoding "b is a son of a"). We focused on trees of 50 nodes and a depth of 6. As opposed to 3-CNF case, the number of rewritings in each peer is now quadratic (it corresponds to the number of paths in the tree rooted at the node labelled by the queried literal). We chose $p = 25$ for each

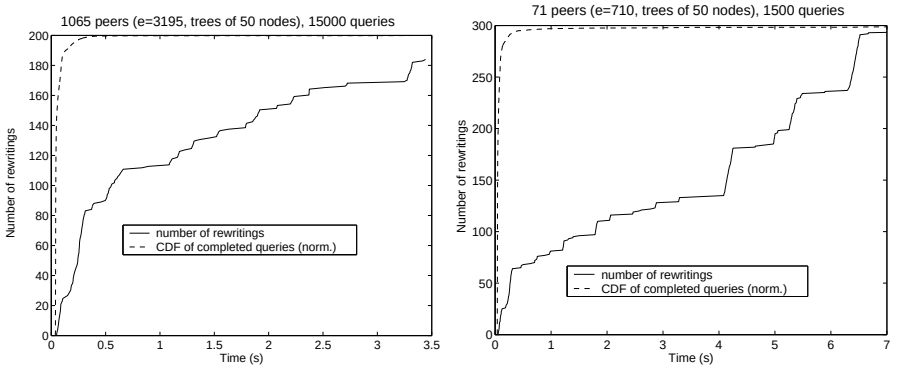

Fig. 4. Left: 1065 peers and $c = 3$ – Right: 71 peers and $c = 10$

d	n	Completion time	Rewriting heterogeneity	Rewriting length
1065	3195	0.06 (0.17) [0.04 0.13]	1.45 (2.09) [1 4]	3.63 (10.31) [1 15]
71	710	0.20 (2.94) [0.04 0.19]	1.49 (2.36) [1 4]	4.37 (16.25) [1 18.5]

Table 1. Experimental results for two configurations (columns d and n) with q=2, m=50, n=50 (CNF encoding of trees). The numbers are: mean (std) [median 95%]. Time is in (real) seconds.

peer. Additional clauses relate variables shared by connected peers, randomly stating equivalence or implication. Two kinds of benchmarks have been considered. Results are summerized on figure 4 (in a similar way to figure 3) and on table 1.

The first benchmark focuses on a large number of peers ($d = 1065$, $e = 3195$ and $c = 3$). The first striking point is that all queries terminate in less than 3.5s, (95% of them in less than 0.04s). Moreover, the mean number of peers composing rewritings is 1.45 (max=37). Although the curves are not presented here, we observe an exponential distribution of the number of distinct peers contributing to rewritings. On more than 5% of the queries, rewritings involve more than 4 different peers.

The second benchmark considers a smaller network with a higher connection ratio ($d = 71$, $e = 710$, $c = 10$), supposed to be more representative of a *small world* (i.e., a community of users sharing points of interest) . Each peer has 40% of its variables in a mapping. All the queries terminated in less than 7 seconds (95% in less than 0.19 seconds). We observed a similar heterogeneity of the rewritings which are a bit longer.

We think that acquaintance graphs of more realistic applications should correspond to large graphs of weakly connected small worlds. The above results suggest that SomeWhere might scale up. Further experiments will be done to validate this hypothesis.

6 Related work & conclusion

Peer-to-peer systems have evolved from keyword-based file sharing systems like Gnutella (www.gnutella.com) and KaZaA (www.kazaa.com) to semantic peer data management systems like Piazza [9], Edutella [10] and SomeWhere.

Piazza has XML as data model. Mappings (relying on a subset of XQuery) are inclusions and equivalences between documents, which makes Piazza and SomeWhere architectures quite similar. As in SomeWhere, the main focus of Piazza is to compute all the answers of a query by rewriting queries using views. A small but realistic Piazza

application had been deployed in order to obtain first experimental results. The application relates 15 peers concerning different aspects of the database research field. Five test queries are given with their respective rewriting times and number of rewritings.
Edutella has RDF as data model and its architecture significantly differs from Piazza and SomeWhere. First, peers do not play the same role within the network. Some of them are just data providers. Others are wrapping mediators that distribute queries to appropriate peers, each of which can answer locally the queries it receives. More complex peers are integrating mediators that are able to mediate distributed queries over multiple peers. The other difference comes from the mappings which are not declared between peers. Each peer registers the queries it is able to answer by providing to some mediator peers information about its schema, constraints and query language.

The contributions of this paper is the SomeWhere system: a peer-to-peer architecture for the OWL W3C emerging standard. First experiments clearly illustrate its ability to scale up to hundreds of peers. On structured theories, we tested it with more than a thousand peers and more than fifteen thousands queries. All the queries ended in less than four seconds, giving rewritings built on multiple peers vocabulary. This architecture is used in a joint project with France Télécom, which aims at enriching peer-to-peer web applications with reasoning services (e.g., Someone [11]).

We plan to extend SomeWhere in two directions. First, we want to take into account a larger fragment of OWL DL. This will help in managing and querying more complex data within a SomeWhere network. We also want to make SomeWhere more robust and dynamic by using a Chord-like network layer [12] in order to manage the connections, disconnections and look-up services of peers.

References

1. Berners-Lee, T., Hendler, J., O.Lassila: The semantic web. Scientifi c American, 279 (2001)
2. Halevy, A., Z.Ives, D.Suciu, I.Tatarinov: Schema mediation in peer data management systems. In: Proceedings of ICDE. (2003)
3. http://www.w3.org/TR/owl semantics: Owl web ontology language semantics and abstract syntax. Technical report, W3C (2004)
4. Goasdoué, F.: Réécriture de requêtes en termes de vues dans CARIN et intégration d'informations. PhD thesis, Université Paris Sud XI - Orsay (2001)
5. Goasdoué, F., Rousset, M.C.: Answering queries using views: a krdb perspective for the semantic web. ACM Journal - Transactions on Internet Technology (TOIT) **4** (2004)
6. Adjiman, P., Chatalic, P., Goasdoué, F., Rousset, M.C., Simon, L.: Distributed reasoning in a peer-to-peer setting. Technical Report 1385, Université Paris Sud XI (2004)
7. Schrag, R., Crawford, J.: Implicates and prime implicates in random 3-sat. Artifi cial Intelligence **81** (1996) 199–221
8. Simon, L., del Val, A.: Effi cient consequence fi nding. In: IJCAI'01. (2001) 359–365
9. Halevy, A., Ives, Z., Tatarinov, I., Mork, P.: Piazza: data management infrastructure for semantic web applications. In: WWW'03. (2003)
10. Nedjl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmer, M., Risch, T.: Edutella: a p2p networking infrastructure based on rdf. In: WWW'02. (2002)
11. Plu, M., Bellec, P., Agosto, L., van de Velde, W.: The web of people: A dual view on the WWW. In: Int. World Wide Web Conf. (2003)
12. Stoica, I., Morris, R., Karger, D., Kaasshoek, M., Balakrishnan, H.: Chord: A scalable Peer-To-Peer lookup service for internet applications. In: ACM SIGCOMM. (2001) 149–160