Distributed Datalog Revisited*

Serge Abiteboul[1], Meghyn Bienvenu[2],
Alban Galland[1], and Marie-Christine Rousset[3]

[1] INRIA Saclay & ENS Cachan,
firstname.lastname@inria.fr,
[2] Univ. Bremen
meghyn@informatik.uni-bremen.de
[3] Univ. Grenoble
Marie-Christine.Rousset@imag.fr

1 Introduction

The emergence of Web 2.0 and social network applications has enabled more and more users to share sensitive information over the Web. The information they manipulate has many facets: personal data (e.g., pictures, movies, music, contacts, emails), social data (e.g., annotations, recommendations, contacts), localization information (e.g., bookmarks), access information (e.g., login, keys), web services (e.g., legacy data, search engines), access rights, ontologies, beliefs, time and provenance information, etc. The tasks they perform are very diverse: search, query, update, authentication, data extraction, etc. We believe that all this should be viewed in the holistic context of the management of a distributed knowledge base. Furthermore, we believe that datalog (and its extensions) forms the sound formal basis for representing such information and supporting these tasks. In this paper, we revisit datalog with this goal in mind. The focus of the presentation is on the formal extension of the model of distributed datalog and does not consider the implementation or the evaluation of the corresponding system [7].

We use logical (datalog) statements to capture these different facets of information that are typically considered in isolation. Knowledge can be communicated, replicated, queried, updated, and monitored. The use of a formal model allows information to be obtained by performing complex reasoning. Our model encompasses a rich variety of scenarios ranging from information in centralized servers to massively distributed, from fully trusted to untrusted, and possibly encrypted information, thereby capturing the reality of today's Web. It also provides the possibility of formally proving or disproving desirable properties such as soundness (data is only acquired legally) and completeness (one can acquire all data that one can legally claim).

After some preliminaries in Section 2, we introduce the model in Section 3. We briefly mention extensions in Section 4. The last section is a conclusion.

* This work has been partially funded by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant Webdam, agreement 226513. http://webdam.inria.fr/

2 Preliminaries

In this section, we first consider the alphabets we use, then the kind of knowledge we manipulate. Finally, we briefly recall distributed datalog.

Alphabets A central notion is that of a *principal* that corresponds to a participant in the system. Our terminology is motivated by the notion of "principal" in the domain of security, i.e., an entity that can be identified and verified via authentication. In the same spirit, a principal is determined in our context by a URI (and possibly authentication keys). Some principals, the *peers*, have a physical address, e.g., Facebook, Alice's iPhone, with storage and processing capabilities. Others may represent a user, e.g., Alice, or a community such as Alice's friends or the rock climbing group.

The information of a principal p is organized in relations. The identification of a relation is of the form $r@p$ where r is the relation name and p the principal. For instance, one might have the relation *pictures*@*alice-iPhone* (the pictures album stored on Alice's iPhone) or the relation *expert*@*rockClimbing* (the experts known in this group).

One can query a peer since it has a URI that corresponds to a real system. For instance, one can obtain the relation *pictures*@*alice-iPhone* by accessing the iPhone (assuming one has access to it). On the contrary, one cannot query a non-peer principal. For instance, one cannot ask *rockClimbing* (a virtual entity) for the list of experts. To obtain such information, one needs rules that tell us how to get information, e.g. for the *rockClimbing* experts, typically by querying "real" relations (extensional or intentional) at some peers.

More formally, the model uses the following alphabets:

- A set $\mathcal{P}'$ of *principal* IDs, that includes a set $\mathcal{P}$ of *peer* IDs. The system provides a unique IDs for each different principal, in the spirit of the standard notion of URIs.
- A set $\mathcal{R}$ of *relation* IDs. An actual relation name is a pair $r@p$ where r is a relation ID and p is a principal ID.
- A set $\mathcal{D}$ of *constants*. It is the disjoint union of the set of principal IDs, relation IDs and a set of data constants. A data constant is some sequence of bits: e.g., a string, a file (picture, music), an XML document. Principal and relation IDs are also constants so that we can reason about them. Constants are typed, e.g., principal, relation, string, integer, dates, etc.
- A set of *variables*. Similarly to constants, variables are typed. We use words starting with small letters for constants and with capitals for variables.

Knowledge The architecture is illustrated in Figure 1. Consider for instance Peer 2. It has data and rules defining personal relations $r1, r2, r3$. It also has data and rules about relation s of principal q_2 (shared with Peer 1) and about $r@q_1$ (shared with Peer 3). Observe the distinction for each peer, between its local schema (e.g., r_i for Peer 2) and its participation in the global schema (e.g., $r@q_1, s@q_2$ for Peer 2).

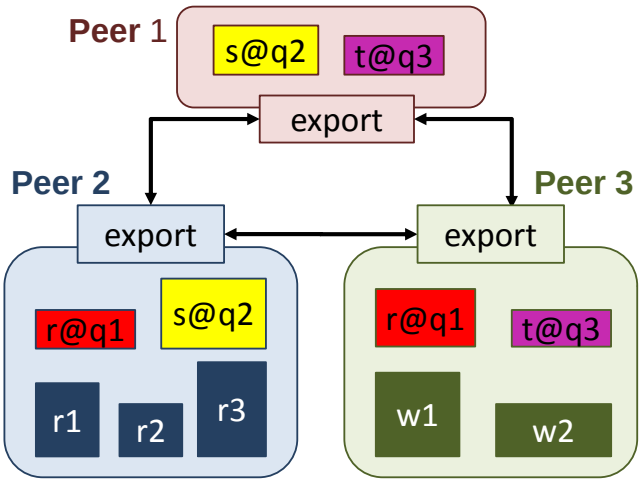

Fig. 1. Distributed datalog framework

Basic knowledge is expressed using facts of the form $r@p(a_1, ..., a_n)$. Note that peers and relations are reified[4]. It is therefore possible to have a fact $r@p(r', p')$ that speaks of a relation r' and a peer p'. Furthermore, we can use peer and relation variables, e.g., $r@p(R, P)$ with R a relation variable and P a peer variable. From a formal viewpoint, we can see each fact $r@p(a, b)$ in a binary relation $r@p$ as a fact $univ_4(r, p, a, b)$ in a 4-ary relation. Indeed, we can even logically see it as a fact $univ(r, p, a, b, -, ..., -)$ in a unique relation, wide enough to accommodate all the tuples that are considered. Even if we introduced a more readable notation for our model, the reader should keep in mind that our model can be fully described in terms of standard datalog.

In a distributed context, a peer p is naturally led to possess data about other participants, for instance a principal who uses p as server for his data, or another peer that replicates its data on p to offer better performance or availability. For instance, consider the fact $album@alice(a)$. This fact may be known, e.g., by two peers, Picasa and Alice's iPhone. The knowledge of this fact by these peers, respectively, can be stated as:

$$picasa\ \ exports\ \ album@alice(a)$$
$$alice\text{-}iPhone\ \ exports\ \ album@alice(a)$$

The use of the *exports* word is motivated by the fact that one can now query Picasa or Alice's iPhone, to obtain such data, i.e., this data is *exported* by these two peers. Indeed, the primary goal of such a modality is to capture the essence of

[4] Reification is a process through which a computable/addressable object acts as a proxy for a non computable/addressable object.

communications between peers. For instance, we can ask such data from Picasa using (informally) the query: *picasa exports album@alice*(X).

More formally, we model such knowledge using a particular relation called *export*. The fact that some peer p *exports* $r@q(a_1, ..., a_n)$ is stated as:

$$export@p(r, q, a_1, ..., a_n).$$

Note that the use of special relations such as *export* (or *univ* above) complicates typing. Suppose for instance that we store in relation $export@p$ tuples in $r@q_1$ and $r@q_2$ and that the first contains pairs of peers and the second data elements. Then the "type" of a tuple in $export@p$ is: $\langle r, q_1, peer, peer\rangle$ or $\langle r, q_2, data\rangle$. The use of these untyped relations may be seen as syntactic sugaring for some relations of distinct types.

As already mentioned, the relation *export* captures communications (interactions) between the peers. It therefore plays an essential role in the model. Particular applications may use other specific relations with particular associated semantics. For instance, one could consider:

- Localization: e.g., $where@q(r, p)$. Such a fact in the q principal indicates that relation $r@q$ can be obtained from peer p.
- Access rights: $access@p(read, q)$. This fact states that q has read access to principal p.

We will use such relations in examples to illustrate the model.

Classical distributed datalog We briefly present distributed datalog which is a rather straightforward variation of standard datalog. We refer to, e.g., [4], for technical details on datalog. To our knowledge, the first attempts to distribute datalog on different peers are [15] and [18]. Since ten, many works have followed. For instance, one of the authors of this article adapted query-sub-query optimization to a distributed setting in [1]. Distributed versions of datalog have also recently been used to implement Web routers [17], DHT [16] and Map-Reduce [6] very efficiently. These works only consider peer principals, since the focus is on the exchange of data between machines.

We present here a standard version of Distributed Datalog inspired by [1]. To illustrate, the following program defines the album of Alice's iPhone to include all the photos stored in her iPhone (a local relation) and on Picasa (a relation on another peer):

$$album@alice\text{-}iPhone(X) \leftarrow photos@alice\text{-}iPhone(X)$$
$$album@alice\text{-}iPhone(X) \leftarrow aliceAlbum@picasa(X)$$

Precisely, a *DDL* (for distributed datalog) *schema* is a triple (Π, Rel, σ) where Π is a finite set of peer IDs, for each $p \in \Pi$, $Rel(p)$ is a finite set of relation IDs, and for each p, r, $\sigma(r@p)$ specifies the type of relation $r@p$, i.e., its arity and the types of its columns.

A *DDL instance* I of (Π, ρ, σ) maps each p to a finite set $I(p)$ of "safe" datalog *rules* defining relations in p.

We will formalize the notion of safety when we describe *distributed datalog revisited*. Rules with an empty body, e.g., $r@p(5) \leftarrow$, are called *facts*. Observe that, by definition, a rule of p cannot have in its head a relation at peer p'. We will relax this restriction in Remark 3.

Remark 1. Observe that a rule at peer p may use in its body a relation at peer p'. This may be realized in practice by posing a query to p'. The query is continuous in the sense that p' keeps sending matching tuples to p as long as it derives them. We will ignore here the issue of detecting termination, which may be performed using some classical techniques.

A *query at (peer) p* is a rule of the form:

$$query@p(u) \leftarrow r_1@p_1(u_1), ..., r_n@p_n(u_n)$$

Let I be an instance. The semantics $close(I)$ is defined as the smallest set of facts over $\cup_p Rel(p)$ that satisfies $\cup_p I(p)$.

We assume here to simplify that each peer knows how to query other peers and that it has access to all the data in other peers. This assumption may be relaxed in some contexts and in particular, when considering access rights.

Note that such specifications are very static, in particular with respect to localization. Also there is the assumption that each principal corresponds to a physical machine and is available all the time. The work presented here removes such limitations.

3 Distributed datalog revisited

In this section, we extend datalog, in order to capture the kind of knowledge one needs to handle in distributed information system. We introduce two extensions, one based on the reification of relations and peers, and the second on the notion of principal.

Reifying relations and peers The first extension we consider is based on reifying relations and peers. (We still ignore for now non-peer principals.) To illustrate, suppose that Alice is storing her album on some machine and Bob does not know where. Suppose also their group of friends decided on a global localization service, say *directory@server*. Then Bob may use the following rule:

$$\begin{aligned} \textit{alice-album}@\textit{bob-iPhone}(X) \leftarrow\ & \textit{directory}@\textit{server}(\textit{album},\textit{alice},P), \\ & \textit{export}@P(\textit{album},\textit{alice},X) \end{aligned}$$

Here P, R are respectively peer and relation variables. The service *directory@server* provides bindings for P (the name of the system) and R (the name of the relation on this system) where Alice's album can be found. Observe that similar use of localization services are standard in distributed contexts, e.g., DHT or Ldap.

Non-peer Principals We extend the notion of schema, instance, and their semantics to now include non-peer principals. To illustrate, assume that Alice, Bob and others created a principal *rockClimbing*. They may want to use a relation *album* in this shared principal. Bob may for instance use the rules:

$$album@rockClimbing(X) \leftarrow album@bob\text{-}iPhone(X)$$
$$album@rockClimbing(X) \leftarrow friends@bob\text{-}iPhone(Q),\ directory@server(album,Q,P),\ export@P(album,Q,X)$$

Now observe that *album@rockClimbing* is a "global" relation and that many peers may "publish" data in it and many peers may know how to retrieve data from it. The second rule highlights three main ways of obtaining information:

- $friends@bob\text{-}iPhone(Q)$ matches some facts in a (local) relation of this particular peer;
- $directory@server(album, Q, P)$ matches some facts in a relation of another peer;
- $export@P(Q, album, X)$ matches some facts exported by another peer (i.e., external knowledge of that peer).

The difference between the second and third kind of atoms is that the second one is a form of syntactic sugaring which hides the explicit communication step. Indeed, from a communication point of view, the evaluation of $directory@server(album, Q, P)$ results in a query to *server*, namely:

$$export@server(directory, server, album, Q, P)$$

There is yet a fourth kind of atom of the form $t@q(\bar{V})$ that matches the peer's knowledge of some non-peer principal relation $t@q$.

The example illustrates a key usage of our model for maintaining "external knowledge". Suppose that Alice is a friend of Bob (Q is bound to Alice). Intuitively, we would like to get the album of Alice. What we query instead is a machine say Picasa (P is bound to Picasa) where Alice stores her pictures.

Formal model A *term* is a constant or a variable. In particular, a relation term, a peer term and a principal term are terms of type relation, peer and principal respectively.

A *DDLR* (for distributed datalog revisited) *rule* is an expression of the form:

$$\alpha_0@\beta_0(V_0) \leftarrow \alpha_1@\beta_1(V_1), ..., \alpha_n@\beta_n(V_n)$$

where

- α_i, β_i are relation and principal terms respectively.
- V_i are tuples of terms.
- The rule is *safe* in that (i) in the body, a principal or relation variable must be bound before appearing as α_i or β_i, and (ii) a variable occurring in the head must be bound in the body.

Note the ordering of atoms in the previous definition. This can be relaxed by imposing that for *some* ordering of the atoms, (i) and (ii) hold.

A *DDLR schema* is a quadruple (Π', Π, Rel, σ) where Π is a set of peer IDs, $\Pi' \supseteq \Pi$ a set of principals, Rel assigns relation IDs to principals, and σ gives their arities.

A *DDLR instance* I of (Π', Π, Rel, σ) maps each $p \in \Pi$ (each peer) to a finite set $I(p)$ of safe DDLR rules such that:

(head) The relation in the head is (+) either $R@p(\bar{V})$ for some relation term R; or (++) $R@Q(\bar{V})$ for some relation term R and *non-peer* principal term Q.
(body) Each relation in the body is either of the form $R@Q(\bar{V})$ for some relation term R and some (peer or not) principal term Q or of the form $export@P(\bar{V})$ for some peer term P.
(typing) Each rule respects the typing specified by the schema.

Henceforth, unless otherwise specified, datalog means DDLR.

Semantics There are subtleties in the inference of facts. First, note that the classical least-model semantics, i.e. $close(\cup_p I(p))$ is in general not attainable. For instance, suppose peer p has a rule $r@p(x) \leftarrow s@q(x)$ where q is a non-peer principal and has no rule defining q. Then p cannot call q so this rule is useless, even if q is defined on another peer.

Observe another subtlety: (+) uses typing to prevent the derivation of information in other peer relations. We will show how to relax (+) in Remark 3.

For a set K of facts, a relation r and a principal q, $K(r@q)$ is the sets of facts in K about $r@q$.

Definition 1. *(Distributed least-model semantics) Let I be an instance. The* dclosure *is a function (i) that maps each peer p, to a set $dclose(I,p)$ of facts over $\cup_q Rel(q)$ (facts over peers and non-peer principals) and (ii) that satisfies:*

1. *for each pair of peers p, p' and each r, $dclose(I,p)(r@p) = dclose(I,p')(r@p)$.*
2. *for each peer p, $dclose(I,p)$ satisfies $I(p)$.*
3. *$\cup_p dclose(I,p)$ is minimum.*

Observe that all peers have the same view of the peer relations. Suppose peer p does not even know of peer p' and relation r' in it, i.e., p' does not occur in its local knowledge. By the previous definition, $dclose(I,p)(r'@p') = dclose(I,p')(r'@p')$. This may seem unnatural. However, suppose someone asks the query $\leftarrow r'@p'(x_1, ..., x_n)$ to p. Then p discovers the existence of $r'@p'$, can get data from p' and can answer the query. This motivates (1) in the previous definition.

Note that, in general, the different views the peers have of the principal relations may be incomplete. It is easy to see that in general, for a peer p_0:

$$dclose(I,p_0) \subseteq \cup_p dclose(I,p) \subseteq close(\cup_p I(p))$$

and that the inclusions are possibly strict. Note that this leads to complex reasoning about knowledge in the style of [12]. One may want to check whether a

given set of rules guarantees completeness. Unfortunately, verifying such properties comes down to comparing datalog programs, which is undecidable [10].

We conclude this section with two remarks: one on open vs. closed world assumptions, and one on relaxing (+) by allowing rules with other peer relations in the head.

Remark 2 (Open vs. closed world). It should be observed that such definitions of "global principal relations" typically rely on an open-world assumption since anyone (with proper access right) can participate in the definition of that relation and perhaps no one has the complete picture. This is in the spirit of Local-as-View mediator systems [13]. On the other hand, consider a peer relation. The value of that relation at the peer can be seen as its complete instance, so it is more in the spirit of the closed world assumption and Global-as-View. Clearly, both kinds of relations may be combined freely. For instance, a definition of a peer relation may use a non-peer principal relation.

Remark 3 (Relaxing (+)). Consider a peer p_1 with the rule $r@p_2(x) \leftarrow s@p_1(x)$. An issue is that when asked the query $\leftarrow r@p_2(X)$, peer p_2 may not be aware of the rule at p_1. We could allow such a rule if (intuitively), p_1 notifies p_2 that he may participate to the definition of $r@p_2$ and p_2 reacts by installing locally the rule:

$$r@p_2(X) \leftarrow \mathit{export}@p1(r, p2, X)$$

that may be interpreted by "p_1 also has facts for $r@p_2$".

4 Extensions in brief

A holistic knowledge-base model would also need to rely on a number of extensions of datalog that have so far been studied in isolation. We mention them next. Datalog has to be extended in the following ways:

Nonmonotonicity. One can first consider negation in rules with different semantics, e.g., stratified negation or well-founded negation, see [4]. Problems of nonmonotonicity also happen when one considers updates, that are necessary to capture real applications. One could also consider negation in heads of rules. So for instance, someone may state that Bill is not an expert in rock climbing. This may contradict the statement of someone else who states that he is. Clearly, such inconsistencies are frequent on the Web and a comprehensive model for Web data management should take this into account.

Ontologies and incomplete information. Ontologies can be used to structure a participant's vocabulary and to translate knowledge between the vocabularies of different participants in a distributed environment, cf. e.g., [19]. Some simply ontology statements, like predicate inclusions (e.g. $\mathit{Photo} \sqsubseteq \mathit{Document}$), can be straightforwardly handled by our proposed framework. However, other important ontological constructs, like existential restrictions ($\mathit{Parent} \sqsubseteq \exists \mathit{hasChild}$) which may introduce incomplete information, are not supported. Extensions of datalog in this direction have been considered, see [9].

Intentional data. We assumed in this paper that we answer queries with facts. It may be appropriate to answer with "rules". For instance, if one asks a machine for a relation, say R, in the *rockClimbing* principal, it may be preferable to obtain as answer, some rule for computing R, e.g., stating that one should first obtain some data from Bob and combine it with data from Alice. This presents the advantage of allowing learning about new relations, a feature typically needed on the Web. Extensions of datalog in this direction have been considered under the generic term of Active XML [2, 5].

Trees. The data exchange format of the Web is XML, i.e. trees instead of relations. Active XML also extends datalog to trees.

Time. When we consider evolving data, time becomes an issue since a relation at time t may be different than at time t'. Extensions of datalog with time have been studied, e.g. in [14]. The idea is to distinguish between relations $r@t$ and $r@t'$, in the same spirit that we thought of the knowledge of *rockClimbing* as different between two sites $s1$ and $s2$.

Access rights. Access rights have been considered in this setting in the WebdamExchange system [3]. For instance, we can augment the *export* relation with an extra column for the identity of the caller. Then to query a peer p, we use this extra column, e.g.,
$exports@picasa(album,lulu,X,alice)$.
Alice uses this query (properly authenticated with her signature) for asking pictures in Lulu's album from Picasa. Based on the access right of Alice, Picasa chooses which data to return. Due to space limitations, this will not be detailed here.

Beliefs. Peers or principals may want to state information they believe in but are not sure. This may be captured, for instance, by extending the *export* relation with yet one more column. The fact
$export@server(where,\ album,\ alice,\ bobIPhone,\ 72\%)$
states that the server believes that the album of Alice can be found on Bob's iPhone with a strong probability (72%).

5 Conclusion

Distribution leads naturally to recursive query processing. An entry point to the field may be found in [4]. Optimization techniques for datalog have been developed, e.g., QSQ [20] and Magic Set [8]. There have been many works on optimization techniques for distributed and recursive query processing, e.g., semi-join techniques to minimize communications [18]. QSQ-like optimizations with data flow evaluations and a termination detection mechanism are considered, e.g., in [15]. QSQ has been adapted to a distributed setting in [1].

There has recently been renewed interest in datalog; see [11]. We presented a knowledge model for the management of distributed (Web) information based on datalog. We already discussed a number of extensions in Section 4, so possible directions for future works. Some of the authors are currently developing a system based on distributed datalog revisited, namely WebdamExchange, with a strong emphasis on access control [3].

The work presented here raises a number of issues. In particular, we need to investigate the reasoning needed to find information of interest or disseminate such information. A main issue is the efficient support of queries in systems based on distributed datalog. Optimization techniques such as QSQ [20] have been adapted to similar setting, e.g., [1]. Also, it would be interesting to study properties of the resulting systems such as completeness, i.e., any available data can be obtained. In the context of access rights, completeness takes a new flavor: "any data one is entitled to see can be obtained". Also, this raises the issue of *soundness*: "only data one is entitled to see can be obtained".

References

1. S. Abiteboul, Z. Abrams, S. Haar, and T. Milo. Diagnosis of asynchronous discrete event systems: datalog to the rescue! In *PODS*, pages 358–367, 2005.
2. S. Abiteboul, O. Benjelloun, and T. Milo. The Active XML project: an overview. *VLDB J.*, 2008.
3. S. Abiteboul, A. Galland, A. Marian, and A. Polyzotis. A model for web information management with access control. In preparation, 2010.
4. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
5. S. Abiteboul, L. Segoufin, and V. Vianu. Static analysis of Active XML systems. In *PODS*, pages 221–230, 2008.
6. Peter Alvaro, Tyson Condie, Neil Conway, Khaled Elmeleegy, Joseph M. Hellerstein, and Russell Sears. Boom analytics: exploring data-centric, declarative programming for the cloud. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 223–236, New York, NY, USA, 2010. ACM.
7. E. Antoine, A. Galland, K. Lyngbaek, A. Marian, and N. Polyzotis. Social networking on top of the WebdamExchange system. In *ICDE*, 2011 (To appear).
8. F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic sets and other strange ways to implement logic programs. In *PODS*, pages 1–15, 1986.
9. A. Calì, G. Gottlob, and T. Lukasiewicz. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *ICDT*, pages 14–30, 2009.
10. S. S. Cosmadakis, H. Gaifman, P. C. Kanellakis, and M. Y. Vardi. Decidable optimization problems for database logic programs (preliminary report). In *STOC*, pages 477–490, 1988.
11. Datalog 2.0. http://www.datalog20.org/, 2010. Oxford Univ.
12. R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning about knowledge*. The MIT Press, 2003.
13. A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.
14. J. Hellerstein. The declarative imperative: Experiences and conjectures in distributed logic. *SIGMOD Rec.*, 39:5–19, September 2010.
15. G. Hulin. Parallel processing of recursive queries in distributed architectures. In *VLDB*, pages 87–96, 1989.
16. Boon Thau Loo, Tyson Condie, Joseph M. Hellerstein, Petros Maniatis, Timothy Roscoe, and Ion Stoica. Implementing declarative overlays. *SIGOPS Oper. Syst. Rev.*, 39:75–90, October 2005.

17. Boon Thau Loo, Joseph M. Hellerstein, Ion Stoica, and Raghu Ramakrishnan. Declarative routing: extensible routing with declarative queries. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '05, pages 289–300, New York, NY, USA, 2005. ACM.
18. W. Nejdl, S. Ceri, and G. Wiederhold. Evaluating recursive queries in distributed databases. *IEEE Trans. Knowl. Data Eng.*, 5(1):104–121, 1993.
19. M.-C. Rousset, P. Adjiman, P. Chatalic, F. Goasdoué, and L. Simon. Somewhere in the semantic web. In *SOFSEM*, pages 84–99, 2006.
20. L. Vieille. Recursive axioms in deductive databases: The query/subquery approach. In *Expert Database Conf.*, pages 253–267, 1986.