Computer Languages, Systems & Structures 38 (2012) 158–180

Contents lists available at SciVerse ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Maintaining distributed logic programs incrementally

Vivek Nigam [a,*], Limin Jia [b], Boon Thau Loo [c], Andre Scedrov [c]

[a] Ludwig-Maximilians-Universität, Germany
[b] Carnegie Mellon University, USA
[c] University of Pennsylvania, USA

ARTICLE INFO

Article history:
Received 27 October 2011
Received in revised form
8 February 2012
Accepted 10 February 2012
Available online 17 February 2012

Keywords:
Declarative Networking
Correctness
Logic Programming
Distributed Datalog

ABSTRACT

Distributed logic programming languages, which allow both facts and programs to be distributed among different nodes in a network, have been recently proposed and used to declaratively program a wide-range of distributed systems, such as network protocols and multi-agent systems. However, the distributed nature of the underlying systems poses serious challenges to developing efficient and correct algorithms for evaluating these programs. This paper proposes an efficient asynchronous algorithm to compute incrementally the changes to the states in response to insertions and deletions of base facts. Our algorithm is formally proven to be correct in the presence of message reordering in the system. To our knowledge, this is the first formal proof of correctness for such an algorithm.

1. Introduction

One of the most exciting developments in computer science in recent years is that computing has become increasingly distributed. Both resources and computation no longer reside in a single place. Resources can be stored in different machines possibly around the world, and computation can also be performed by different machines, *e.g.* cloud computing. Since machines usually run asynchronously and under very different environments, programming computer artifacts in such frameworks has become increasingly difficult as programs have to be at the same time correct, readable, efficient and portable. There has, therefore, been a recent return to using declarative programming languages, based on Prolog and Datalog, to program distributed systems such as networks and multi-agent robotic systems, *e.g.* Network Datalog (*NDlog*) [10], MELD [5], Netlog [6], DAHL [12], Dedalus [4]. When programming in these declarative languages, programmers usually do not need to specify *how* computation is done, but rather *what* is to be computed. Therefore declarative programs tend to be more readable, portable, and orders of magnitude smaller than their imperative counterparts.

Distributed systems, such as networking and multi-agent robotic systems, deal at their core with maintaining states by allowing each node (agent) to compute locally and then propagate its local states to other nodes in the system. For instance, in routing protocols, at each iteration each node computes locally its routing tables based on information it has gained so far, then distributes the set of derived facts to its neighbors. We can specify these systems as distributed logic programs, where the base facts as well as the rules are distributed among different nodes in the network.

Similar to its centralized counterparts, one of the main challenges of implementing these distributed logic programs is to efficiently and correctly update them when the base facts change. For distributed systems, the communication costs due to updates also need to be taken into consideration. For instance, in the network setting, when a new link in the network has been established or an old link has been broken, the set of derived routes need to be updated to reflect the changes in the base facts.

* Corresponding author. Tel.: +49 89 21 80 93 37.
E-mail addresses: vivek.nigam@ifi.lmu.de (V. Nigam), liminjia@cmu.edu (L. Jia), boonloo@cis.upenn.edu (B.T. Loo), scedrov@math.upenn.edu (A. Scedrov).

1477-8424/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cl.2012.02.001

It is impractical to re-compute each node's state from scratch when changes occur, since that would require all nodes to exchange their local states including those that have been previously propagated.

A better approach is to maintain the state of distributed logic programs incrementally. Instead of reconstructing the entire state, one only modifies previously derived facts that are affected by the changes of the base facts, while the remaining facts are left untouched. For typical network protocols, updates to the base facts are caused by topology changes, and these changes are small compared to the size of the entire network, but happen quite often. Therefore, whenever a link update happens, incremental recomputation requires less bandwidth and results in much faster protocol convergence times when compared to recomputation from scratch. (We compare incremental approach to recomputation in more detail at the end of Section 2.3.)

This paper develops algorithms for incrementally maintaining recursive logic programs in a distributed setting. Our algorithms allow asynchronous execution among agents. No agent needs to *stop* computing because some other agent has not concluded its computation. Synchronization requires extra communication between agents, which comes at a huge performance penalty. In addition, we also allow update messages to be received out of order. We do not assume the existence of a *coordinator* in the system, which matches the reality of distributed systems. Finally, we develop techniques that ensure the termination of updates even in the presence of recursive logic programs.

More concretely, we propose an asynchronous incremental logic programming maintenance algorithm, based on the *pipelined semi*-naïve(PSN) evaluation strategy proposed by Loo et al. [10]. PSN relaxes the traditional semi-naïve (SN) evaluation strategy for Datalog by allowing an agent to change its local state by following a local pipeline of update messages. These messages specify the insertions and deletions scheduled to be performed to the agents' local state. When an update is processed, new updates may be generated and those that have to be processed by other agents of the system are transmitted accordingly.

We discovered that existing PSN algorithms [10,9] may produce incorrect results if the messages are received out of order. We propose a new PSN algorithm and formally prove its correctness. Up to our knowledge, this is the first formal proof for such an algorithm under the assumption that messages can be received out of order. What makes the problem hard is that we need to show that, in a distributed, asynchronous setting, the state computed by our algorithm is correct regardless of the order in which updates are processed. Unlike prior PSN proposals [10,9], our algorithm does not require that message channels be FIFO, which is for many distributed systems an unrealistic assumption.

Guaranteeing termination is another challenge for developing an incremental maintenance algorithm for distributed recursive logic programs. Typically, in a centralized synchronous setting, algorithms, such as DRed [7], guarantee the termination of updates caused by insertion by maintaining the set of derivable facts, and discarding new derivations of previously derived facts. However, to handle updates caused by deletion properly, DRed [7] first deletes all facts that could be derived using a deleted base fact, then DRed re-derives any deleted fact that has an alternative derivation. Re-derivation incurs communication costs, which degrade the performance in a distributed setting. This argues for maintaining the multiset of derivable facts, where no re-derivation of facts is needed, since nodes keep track of all possible derivations for any fact. However, termination is no longer guaranteed, as cycles in the derivation of recursive programs allow facts to be supported by infinitely many derivations.

To tackle this problem, we adapt an existing centralized solution [14] to distributed settings. For any given fact, we add annotations containing the set of base and intermediate facts used to derive that fact. These per-fact annotations are then used to detect cycles in derivations. We formally prove that in a distributed setting, the annotations are enough to detect when facts are supported by infinitely many derivations and guarantee termination of our algorithm.

This paper makes the following technical contributions, after introducing some basic definitions in Section 2:

- We propose a new PSN-algorithm to maintain distributed logic programs incrementally (Section 3). This algorithm only deals with distributed non-recursive logic programs. (Recursive programs are dealt in Section 5.)
- We formally prove that PSN is correct (Section 4). Instead of directly proving PSN maintains distributed logic programs correctly, we construct our proofs in two steps. First, we define a synchronous algorithm based on SN evaluations, and prove the synchronous SN algorithm is correct. Then, we show that any PSN execution computes the same result as the synchronous SN algorithm.
- We extend the basic algorithm by annotating each fact with information about its derivation to ensure the termination of maintaining distributed states (Section 5), and prove its correctness.
- We point out the limitations of existing maintenance algorithms in a distributed setting where channels are not necessarily FIFO (Section 6) and comment on related work (Section 7).

Finally, we conclude with some final remarks in Section 8. This is an extended and revised version of the conference paper [15].

2. Distributed Datalog

We present *Distributed Datalog* (*DDlog*), which extends Datalog programs by allowing Datalog rules to be distributed among different nodes. *DDlog* is the core sublanguage common to many of the distributed Datalog languages, such as *NDlog*[10], MELD [5], Netlog [6], and Dedalus [4]. Our algorithms maintain the states for *DDlog* programs.