J. Parallel Distrib. Comput. 103 (2017) 104–112

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Research on semantic of updatable distributed logic and its application in access control

Li Ma [a], Peng Leng [b], Yong Zhong [a,*], Wenyin Yang [a]

[a] School of Electronic and Information Engineering, Foshan University, Foshan, China
[b] Department of Information Engineering, Wuhan Business University, Wuhan, China

HIGHLIGHTS

- Extends U-Datalog to distributed environment but keeps the logic semantic and evaluation method.
- Define update in distributed environment based on non-immediate update semantics.
- Leads an evaluation convenience and advantages in distributed environment.

ARTICLE INFO

Article history:
Received 1 September 2016
Received in revised form
30 November 2016
Accepted 3 December 2016
Available online 3 January 2017

Keywords:
Distributed logic
Updatable distributed datalog
Access control
UD-Datalog

ABSTRACT

The paper presents a distributed logic UD-Datalog whose advantage lies that it extends U-Datalog to distributed environment but still keeps the logic semantic and evaluation method of U-Datalog. The logic presented a new approach to define update in distributed environment based on non-immediate update semantics which distinguishes the language from other distributed datalog. The language is pure declarative and allows us to use top-down and equivalent bottom-up computational evaluation so the already developed techniques for Datalog evaluation can be reused. Firstly, the paper elaborates the syntax and semantic of the logic. Secondly, the evaluation method of the logic is explained. Finally, an application example of the logic in access control of network is discussed which shows the application and expressiveness of the logic.

© 2016 Elsevier Inc. All rights reserved.

1. Introduction

The emergence of Web 2.0 and social networks have attracted a large amount of users to regularly connect, interact and share information with each other, such as twitter, Facebook, LinkedIn, YouTube. However, the massive data and distributed environment of social networks make traditional access control models difficult to implement [16,21,22,20]. So in recent years, logic programming has been proposed as an attractive foundation for distributed programming based on work in declarative networking [9]. And there has been optimism that declarative languages grounded in Datalog can provide a clean foundation for distributed programming, which makes a hit for research of distributed computation using such languages [7,18,19,17,1,10,11,14]. However, in these papers, the semantics is operational and based on a distribution of the program before the execution.

In view of issues with this model, Joseph presents a new model DEDALUS based on an explicit time constructor [13]. But Galland found the semantics of negation together with the use of time in that model rather unnatural. In particular, time is used as an abstract logical notion to control execution steps and the future may have influence on the past. As a consequence, it is difficult to understand what applications are doing as well as to prove results on their language. Galland presents WebdamLog language for the solutions which still is unnatural for its difficult for distinguishing deductive or active rules or update semantics [12]. And the semantics is still operational.

Especially for updatable state, many deductive database systems admit procedural semantics to deal with updates using an assignment primitive. DEDALUS retain a purely logical interpretation by admitting temporal extensions into their syntax and interpreting assignment or update as a composite operation across time steps rather than as a primitive [6]. Many distributed logic languages use hybrid declarative/imperative languages such as Overlog [15] or Boom [5,4] and other [3,2,18,13,23] often clouded our

* Correspondence to: Electronic and Information Engineering School, Foshan University, Foshan, China.
E-mail addresses: molly_917@163.com (L. Ma), lp521@sina.com (P. Leng), zhongyong@fosu.edu.cn (Y. Zhong), cswyyang@163.com (W. Yang).

http://dx.doi.org/10.1016/j.jpdc.2016.12.006
0743-7315/© 2016 Elsevier Inc. All rights reserved.

understanding of the "correct" execution of single-node programs that performed state updates due to the combination of Datalog and imperative constructs.

The paper presents a distributed logic UD-Datalog whose contributions can be summarized as follows:

- The logic extends U-Datalog to distributed environment but still keeps the logic semantic and evaluation method of U-Datalog in centralized environment so a universal logic can be kept.
- The logic presented a new approach to define update in distributed environment based on non-immediate update semantics which distinguishes the language from other distributed datalog, since an immediate update is the main feature of current distributed datalog.
- The language is pure declarative and allows us to use top-down and equivalent bottom-up computational evaluation so the already developed techniques for Datalog evaluation can be reused, which leads an evaluation convenience and advantages comparing to current hybrid declarative/imperative logic languages in distributed environment.

The remainder of the paper will be organized as follows. Section 2 provides preliminaries covering the concepts of U-Datalog and distributed DU-Datalog. Section 3 gives details of our proposed model, which is the logic semantic. Section 4 illuminates the universal evaluation algorithm of UD-Datalog program with an implementation example. An application example of access control is showed in Section 5 and a contrastive analysis is described in Section 6. The paper is concluded in Section 7.

2. Preliminary

2.1. Overview of U-Datalog

U-Datalog is a type of updatable Datalog programming language, in which predicate atoms consist of both updated atoms representing *Insert* and *Remove*, expressed as $\pm p(t_1, t_2, \ldots, t_n)$. U-Datalog language can be considered a special use case of using *Constraint Logic Programming* (CLP), by which updating atoms $\pm p(t_1, t_2, \ldots, t_n)$ as the constraints of the Datalog rules. Except updatable atoms, other Datalog atoms are *Query Atoms* (QA) phase. Two implementation phases of U-Datalog include *Marking* and *Update*. The marking phase completes the variable binding of the transactions and obtains the update sets. The update phase implements the update sets.

Moreover, transactions in U-Datalog follow the rule in Eq. (1) without carrying headers.

$$U_1, \ldots, U_k, \qquad L_1, \ldots, L_m. \tag{1}$$

In Eq. (1), U_i $(0 \leqslant i \leqslant k)$ is a random update atom. L_i $(0 \leqslant i \leqslant m)$ is an arbitrary literal. For any transaction T, we define that a T's query atom set is the query transaction of T. The update set obtained during the marking phase is T's update transaction. Additionally, a pure Datalog program can be considered a special case of U-Datalog.

2.2. Distributed DU-datalog rules

2.2.1. Domain

A domain is the value range of a predicate variable, and we call a discrete finite domain a finite domain. A predicate can be showed as $p(x_1 : t_1, \ldots, x_n : tn)$, where we call $x_1, \ldots, x_n$ the terms of corresponding domain and t_i $(1 \leqslant i \leqslant n)$ are the types of corresponding terms. If domain of type t_i $(1 \leqslant i \leqslant n)$ is a finite domain, we call x_i $(1 \leqslant i \leqslant n)$ a finite term. For simplicity, we omit predicate type and show a predicate as $p(x_1, \ldots, x_n)$.

2.2.2. Predicate types and entity (location) suffix

We assume the last term of each predicate $p(x_1, \ldots, x_n)$ represents the entity of the predicate. For a intuitive view, we express the predicate $p(x_1, \ldots, x_{n-1})@x_n$ which means the predicate $p(x_1, \ldots, x_{n-1})$ on the entity x_n. We use $Loc(p(x_1, \ldots, x_n))$ to represent entity x_n. Similar to Webdamlog [16], an entity is an object having storage and data processing capabilities, which can be either a physical network node or a virtual entity, such as an account in social networks.

2.2.3. Predicate and atoms

We define a constant a term and a variable a term. Let $p(x_1, \ldots, x_{n-1})@x_n$ be a random predicate symbol, with the terms $x_1, \ldots, x_{n-1}$ in its corresponding domain and x_n is the entity suffix. We call $p(x_1, \ldots, x_{n-1})@x_n$ or $\pm p(x_1, \ldots, x_{n-1})@x_n$ an atom formula or atom for short. Predicates includes two types: extensional predicates shown as P_{ext} and intensional predicates shown as P_{int}. Atoms include two types:

1. Update atoms. $\pm p(x_1, \ldots, x_{n-1})@x_n$ that represent the insertion or deletion of corresponding extensional predicates are called update atoms.
2. Query atoms. The atoms except update atoms are called query atoms. We call atom $p(x_1, \ldots, x_{n-1})@x_n$ or its negation $-p(x_1, \ldots, x_{n-1})@x_n$ as a literal. Negations are constrained to extensional predicates, that is $p(x_1, \ldots, x_{n-1})@x_n$ in body of a rule must be extensional predicates. The constraint is the simplest situation of applying negative atoms but it is enough to satisfy with our requirement. For length of the paper, we do not discuss the negative atoms or negative literals further here.

2.2.4. Programs and transactions

Extensional Database (EDB) consists of ground atoms that represent database status. And *Intensional Database* (IDB) consists of the rule below:

$$h@e \leftarrow u_1@e_1, \ldots, u_k@e_k, \qquad p_{k+1}@e_{k+1}, \ldots, p_m@e_m. \tag{2}$$

In Eq. (2), h is a random query atom, $p_i(k+1 \leqslant i \leqslant m)$ is a random query literal, and $u_i(1 \leqslant j \leqslant k)$ is a random update atom.

Transaction (query) T is a kind of rule without head that has the form:

$$u_1@e_1, \ldots, u_k@e_k, \qquad p_{k+1}@e_{k+1}, \ldots, p_m@e_m \tag{3}$$

where $p_i(k+1 \leqslant i \leqslant m)$ is a query literal, and $u_i(1 \leqslant j \leqslant k)$ is a update atom. We call the set of query literal in T as query transaction of T. And call the update set an update transaction brought by execution of transaction T. The paper uses $?-(u_1@e_1, \ldots, u_k@e_k, p_{k+1}@e_{k+1}, \ldots, p_m@e_m)$ to represent a transaction or query. A shortened form representing a transaction with the same suffix is given as follows:

$$\{T_1\}@e_1, \ldots, \{T_m\}@e_m. \tag{4}$$

If all the entities in a transaction T are e, we call the transaction a single transaction on entity e. In Eq. (4), all $T_i(1 \leqslant j \leqslant m)$ can be looked on as a single transaction, the same as traditional transaction. Sometimes, we omit the entity suffix in a single transaction. In addition, we require the rules in a program are safe, which means all variables occurring in the head also appear in the body and any variables in the body must appear in a sub-goal that is not a negative or the variables come from a finite domain. The safe rules guarantee all variables are restricted, which prevent that an unrestricted variable brings rules or facts that cannot be controlled by the database.

Download English Version:

https://daneshyari.com/en/article/4951590

Download Persian Version:

https://daneshyari.com/article/4951590

Daneshyari.com