社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2003－MPS－45 (1)
2003／6／24

NaraViewを用いた分散論理プログラムにおける推論の視覚化

笹倉 万里子、山崎 進
sasakura@momo.it.okayama-u.ac.jp
岡山大学 自然科学研究科

概要

本稿では分散論理プログラムにおける推論の過程を自動並列化コンパイラのための視覚化システム NaraView のプログラム構造ビューを用いて視覚化する。分散論理プログラムにおいて交換されるメッセージは、時刻、メッセージを送受する論理プログラム、メッセージの階層レベルの三つの要素を持っており、それらをプログラム構造ビューの各軸に対応させることで簡単に視覚化できる。

An Application of NaraView to Reasonings for Distributed Logic Programs

Mariko Sasakura and Susumu Yamasaki
Department of Intelligence Computing and Systems
Graduate School of Natural Science and Technology, Okayama University

Abstract

Visualization has played a significant role in understanding the behavior of distributed programs. We propose a visualization that shows message exchanges in distributed logic programs using the Program Structure View in NaraView. The behavior of distributed logic programs is easily visualized using the Program Structure View since the messages have three parametric factors: time, derivation, and hierarchical level.

1 Introduction

Software visualization can clarify the characteristics and behavior of programs. This clarification is especially useful for parallel and distributed programs [1]. NaraView [6] is a software visualization tool that provides two visualizations for parallelizing a program: the Program Structure View and the Data Dependence View. The Program Structure View (PSV) visualizes the structure of a program. The Data Dependence View displays the data dependences in a loop.

The PSV is designed in three-dimensions using the parametric factors of a program. Each dimension corresponds to one of three factors: time sequence, parallelism and loop nesting. In this paper, we make use of the PSV such that it may visualize the behavior of distributed programs by time sequence, parallelism and hierarchical structures in the programs.

We apply this elaborate system to deal with the visualization of reasonings for distributed program environments. Distributed programs consist of logic programs which are different from each other and executed on different processors.

A three-dimensional approach to the visualization is motivated as an application of NaraView:

(1) The first dimension describes sequences of configurations, as time passes and processes change.

(2) The second dimension is concerned with the (spatial or logical) extension of distributed programs.

(3) The third dimension is required for communication histories to form a configuration of communications in distributed programs for some duration.

As a distributed program environment, a distributed logic program is examined as follows.

As discussed in Shepherdson [7], the “negation as failure” rule is well established:

If a proposition A cannot be proved by a theory P ($P \not\vdash A$), then the negated predicate *not* A may be inferable.

It can work in relation to 3-valued logic models. It is applicable to deductive databases to infer *not* A by applying finite searches of the predicate A. The acquisition of the negated predicate *not* A is generally applicable to abduction (as in Kakas et al. [3]), diagnosis, causal theory and so on.

Assuming distributed environments of programs and/or databases, negation as failure is revised to incorporate the idea that negation as failure is performed at each site of the program or the database. In addition, the communications for the negation as failure applications are clearly visualized.

This idea motivates the formulation of a distributed logic program with negation as failure, which is extended from the distributed program without negation (Ramanujam [5]), and the study of the visualization for negation as failure in distributed program environments. A distributed logic program is a network of logic programs, where (1) the reasoning for each logic program is defined, and (2) the negation as failure evoked by each program is formulated throughout the network.

Given the above background, we present a three-dimensional visualization of reasonings for a distributed logic program, which is a network of logic programs. The communication is a reasoning caused by negation as failure through the network where each logic program reasons using the negation as failure through the network. It consists of displays for:

(i) a sequence of configurations,

(ii) a network of logic programs,

(iii) a configuration of negation as failure through the network for some duration.

2 Reasonings in a distributed logic program

2.1 A distributed logic program

We deal with a network of logic programs which contain negation as failure, where negation as failure through the network is formulated and the communications to implement it are visualized. A distributed general logic program (DGLP, for short) is a tuple

$$< P_1, \ldots, P_n > (n \geq 1),$$

where P_i is the general logic program.

A general logic program is a set of clauses of the form $A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n$ ($n \geq m \geq 0$), where A_0, A_1, …, A_m are atoms (positive literals) and *not* A_{m+1}, …, *not* A_n are negations of atoms (negative literals). A_0 is the head of the clause and $A_1, \ldots, A_m$, *not* $A_{m+1}, \ldots$, *not* A_n is its body. A literal is a positive literal or a negative literal.

The goal is an expression of the form $\leftarrow L_1, \ldots, L_n$, where L_1, …, L_n, where L_1, …, L_n are literals. The empty clause containing no head nor body is denoted by $\square$.

The reasonings by SLD resolution and negation as failure for this goal are briefly given below. For the basic treatments, see Lloyd [4].

(1) A goal $\leftarrow A_1\theta, \ldots, A_{i-1}\theta, L_1\theta, \ldots, L_k\theta, A_{i+1}\theta, \ldots, A_m\theta, not\ A_{m+1}\theta, \ldots, not\ A_n\theta$ is derived from a goal $\leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n$ and a (program) clause $A \leftarrow L_1, \ldots, L_k$, where θ is the most general unifier of the atoms A_i ($1 \leq i \leq m$) and A. If a goal reaches $\square$ by SLD resolution and negation as failure (recursively defined as below), we say that the goal succeeds. If a goal cannot reach $\square$ by means of finite applications of SLD resolution and negation as failure, we say that the goal (finitely) fails. In this paper, we deal with only finite failure.

(2) Negation as failure is a rule that states that: A goal $\leftarrow$ *not* A succeeds if a goal $\leftarrow A$ fails, and a goal $\leftarrow$ *not* A fails if a goal $\leftarrow A$ succeeds for a ground atom A (that is, an atom containing no variables). We have a refined negation as failure, originally presented in Eshghi and Kowalski [2]:

  (i) A goal $\leftarrow$ *not* A succeeds if a goal $\leftarrow A$ fails with the atom A in memory,

  (ii) A goal $\leftarrow$ *not* A fails if a goal $\leftarrow A$ succeeds,

  where A is a ground atom, sometimes called an abducible.

We take a rule of "negation as failure through a network" as follows.

(i) A goal $\leftarrow$ *not* A succeeds if a goal $\leftarrow A$ fails for each general logic program with the ground atom A in memory.

(ii) A goal $\leftarrow$ *not* A fails if a goal $\leftarrow A$ succeeds.

EXAMPLE 1 Assume a DGLP $P = < P_1, P_2 >$ such that

$$P_1 = \{p \leftarrow not\ q\},$$
$$P_2 = \{q \leftarrow r\},$$

where p, q and r are atoms (in propositional logic). Because a goal $\leftarrow q$ fails for both the programs P_1 and P_2, we can have a successful derivation for a goal $\leftarrow p$ in P_1, which requires the failing derivation for a goal $\leftarrow q$ in both P_1 and P_2.

2.2 A communication environment

A communication environment for distributed programs consists of two parts: servers which manage message exchanges, and logic programs. A server can connect to the PSV which visualizes the message exchanges in the server. An overview of the environment is as follows.

- Each logic program P_i ($1 \leq i \leq n$), which is a part of a distributed logic program is implemented as an independent program. We call it an *Independent Logic Program* (*ILP*, for short).
- A server *Session* manages message exchanges between *ILPs*. A *Session* knows which *ILP* participates in this *Session* and controls messages to/from other *ILPs*. A *Session* realizes a DGLP.
- There may be more than one *ILP* and *Session* in an environment. An *ILP* can participate in more than one *Session*. A *Session* can consist of more than one *ILP*.
- A process *Reasoning* is a sequence of derivations that begin with a given goal.
- In a *Session*, more than one *Reasoning* can be executed simultaneously.
- A *Session* can visualize a state of *Reasonings* using the PSV with histories of messages.

Messages for distributed logic programs are defined as Table 1.

2.3 Outline of visualization

A history of message exchanges are visualized for a *Session* using the PSV. The PSV is a three-dimensional visualization in which each axis has a different meaning. The x-axis is the time when a message was sent. The y-axis is the derivation to which a message was related. The z-axis denotes the hierarchical structure of messages.

In a *Session*, more than one *Reasoning* can be executed simultaneously. The PSV visualizes their message exchanges in a figure. We extend the PSV to highlight a selected*Reasoning* by coloring the history of messages related to it.

The advantages of the visualization are:

- We can show the state of a *Session*.
- We can know how derivation calls are evoked in a *Reasoning*.
- If there is a derivation that does not send an end message, we can find it in a visualized figure.

3 Implementation for distributed logic programs

The system we have implemented consists of two parts: a *Session* and an *ILP*. A *Session* controls and records messages, and knows which *ILPs* it participates in when a *Reasoning* is performed. An *ILP* has a logic program and it can perform derivations.

A message is represented as a colored cube in the PSV. Its coordinates are defined as: the time, the derivation which sends/receives the message, and the hierarchical level. The three parametric factors of messages are given by a *Session*.

Time: A *Session* has a global clock. The time is set by the clock. The time of a message is determined when the message is sent/received by the *Session*.

Derivation: A derivation is the derivation to/from which the message is sent. To give the feeling of a spatial extension of a network, derivations that are executed on the same it ILP are placed in the same neighborhood.

Hierarchical level: A hierarchical level is calculated by the derivation call tree. The hierarchical level of a message is the depth of the corresponding derivation in the derivation call tree.

We connect messages with lines according to the semantic configuration of messages. Lines are drawn by applying the following rules:

- In the case of an SFAIL message, connect it to all FAIL messages caused by the SFAIL message.
- In the case of an FAILR message, connect it to an SFAILR message that pairs off with the SFAIL message which called the FAILR message.

Table 1: Messages

Message	Sender	Receiver	Comments
SFAIL	*ILP*	*Session*	The start of a network failing derivation. It is sent by a succeeding derivation.
SFAILR	*Session*	*ILP*	The end of a network failing derivation. It is received by a succeeding derivation.
FAIL	*ILP*	*Session*	The start of a failing derivation.
FAILR	*Session*	*ILP*	The end of a failing derivation.

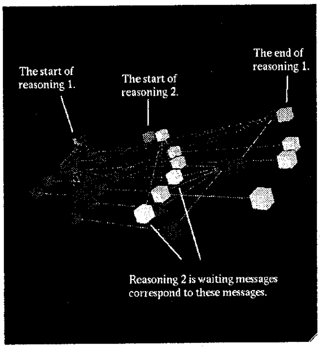

Figure 1: Communications in two *Reasonings* with five *ILPs*.

- In other cases, connect a message to the next message which has the successive time value on the same derivation and hierarchical level.

In the visualization, we add extra cubes that represent the start and the end of a *Reasoning*. The cubes are drawn using a different color than for cubes that represent messages.

EXAMPLE 2 *Figure 1 shows the status of a Session with two Reasonings. The Reasonings have been performed on five ILPs:*$P_1 = \{A \leftarrow not\ B\}, P_2 = \{B \leftarrow C\}, P_3 = \{B \leftarrow not\ D\}, P_4 = \{D \leftarrow not\ E\}, P_5 = \{E \leftarrow \Box\}$ *with a given goal* $\leftarrow A$ *to* P_1 *and a given goal* $\leftarrow B$ *to* P_3. *The former Reasoning finished but the latter has not been finished. The latter Reasoning is depicted in the figure as light gray cubes. The Reasoning is waiting for messages corresponding to the indicated cubes. We can know which derivation is the bottleneck if a Reasoning takes a long time.*

4 Concluding Remarks

We propose a three-dimensional visualization of communications in distributed logic programs with the Program Structure View which is a visualizations in NaraView. The PSV visualizes characteristics of programs on three axes: time, parallelism, and hierarchical structure of programs. In a visualization of reasonings for distributed logic programs, we use derivations as parallelism and the structure of messages caused by recursive calls as the hierarchical structure.

Time and parallelism (derivations, in our case) are common parametric factors to visualize the behavior of parallel/distributed programs. We add a hierarchical structure which is included in the programs as the third axis so that we can get a visualization with a semantic configuration of the programs.

References

[1] P.Eades and K.Chang(eds.). *Software Visualisation.* World Scientific, 1996.

[2] K.Eshghi, and R.A.Kowalski. Abduction compared with negation by failure, *Proc. of 6th ICLP*, 234–255, 1989.

[3] A.C.Kakas, R.A.Kowalski and F.Toni. Abductive logic programming, *J. of Logic and Computation*, 2, 719–770, 1992.

[4] J.W.Lloyd. *Foundations of Logic Programming, 2nd, Extended Edition*, Springer - Verlag, 1993.

[5] R.Ramanujam. Semantics of distributed definite clause programs, *Theoretical Computer Science*, 68, 203–220, 1989.

[6] M.Sasakura, K.Joe, Y.Kunieda and K.Araki. NaraView: an interactive 3D visualization system for parallelization of programs, *International Journal of Parallel Programming*, 27, 2, 111-129, 1999.

[7] J.C.Shepherdson. Negation in Logic Programming, *J.Minker (ed.), Foundations of Deductive Databases and Logic Programming*, 19–88, 1987.