DESIGN AND IMPLEMEMTATION OF DATABASE INTERFACE FOR LOGIC LANGUAGE BASED MOBILE AGENT SYSTEM

Jingbo Ni, Xining Li, Lei Song
Computing and Information Science, University of Guelph, Guelph, Ontario, Canada, N1G 2W1

Keywords: Database Interface, Connection Management, Remote DBMS, Mobile Agent system, IMAGO system

Abstract: Mobile Agent system creates a new way for sharing distributed resources and providing multi-located services. With the idea of moving calculations towards resources, it occupies less network traffics than the traditional Client/Server model and achieves more flexibilities than the Remote Procedure Call (RPC) architecture. In order to endow agents with the ability of accessing remote data resources, in this paper we present the design strategies of the Database Interface between a logic programming language based Mobile Agent system and a remote DBMS.

1 INTRODUCTION

The Mobile Agent system creates a new idea of thin-client design by moving calculations to resource servers. Benefiting from deductive abilities of logic programming languages, complex calculations can be represented by compact logic forms, which make agents more suitable for migrating around. The IMAGO system (Li 2001a & 2001b) is a typical Mobile Agent system based on the IMAOG Prolog (Liang & Li 2003).

Given that the database is one of the most commonly used ways of sharing remote data, this encourages us to discuss possible design strategies of Database Interface between these two systems.

Recently great emphasis has been placed on the relationship between logic programming languages (such as Prolog) and relational databases. Specific SQL string (Applied Logic Systems Inc. 1999; Wielemaker 2002; Bueno et al. 2000), meta-knowledge predicates (Ceri et al. 1989), and variable binding and projection (Draxler 1992) are the three major methods of representing database queries in logic program languages. They are classified as logical *loosely*, *half tightly*, and *tightly* coupled system by Draxler (1992).

Because the evaluations in database systems are set-oriented, as compared to tuple-oriented in logic programming systems, single tuple and set retrieval are the two ways of integrating database relations into logic programming systems. For the single tuple retrieval, results from the database are returned one tuple at a time usually by maintaining a cache or result relations (Mckay et al. 1990), or by using the builtin cursors (Quintus Computer Systems Inc. 1987). For the set retrieval, the entire searching result set is loaded into the program workspace (Ceri et al. 1989) or the running memory layout (Draxler 1992). Educe System (Bocca 1986) offers both of these two strategies.

The searching results can be maintained in the logic language workspace together with other logic facts (Cuppens & Demolombe 1988). Other coupled systems maintain them in local caches (Ceri et al. 1989), temporary buffers or builtin cursors (Quintus Computer Systems Inc. 1987). Also they can be integrated into the running memory in the form of logic lists (Ceri et al. 1989).

Two levels of system efficiency have been introduced: the technical level and the logic level. On the technical level, the Database Connection Management (Mckay et al. 1990) and the Local Cache Management (Sheth & O'Hare 1991) are commonly used. On the logic level, the methods of Delayed Evaluation are presented by Cuppens & Demolombe (1988). Jarke et al. (1984) employed an intermediate language called DBCL between the Prolog and the SQL query for optimising the database search evaluations. Other approaches deal with the logic level efficiency by focusing on the Query Assumption (Nurcan et al. 1990 & 1991).

Ni J., Li X. and Song L. (2005).
DESIGN AND IMPLEMEMTATION OF DATABASE INTERFACE FOR LOGIC LANGUAGE BASED MOBILE AGENT SYSTEM.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 461-464
DOI: 10.5220/0002508604610464

Till now many coupled systems have been developed and among them PROSQL (Chang & Walker 1984), PRIMO (Gozzi et al. 1990), KB-Prolog (Bocca et al. 1989), Nussbaum System (Nussbaum 1988), EKS-VI (Vieille 1990), and the Prolog-SQL coupled by Danielsson & Barklund (1990) are those typical ones.

In this paper we focus on the efficiency issues of the Database Interface design and separate this efficiency problem into three parts: the database connection management, the result memory allocation mechanism and the system memory releasing strategies.

The remaining of the paper is organized as follow. In section 2 we will introduce the IMAGO system. In section 3 the discussion of the Multi-threading Database Connection management is addressed. In section 4 three ways of the Result Memory Pool organization are presented. In section 5 Manual and Automatic System Memory Release methods are described. Conclusions and future works will be given out in section 6.

2 IMAGO SYSTEM

From the computer terminology point of view, the term IMAGO is an abbreviation, which stands for Intelligent Mobile Agents Gliding On-line. Each agent appears in the form of a small logic program written in Prolog and is able to require certain services through a set of builtin predicates (such as agent creation, agent migration, and remote database accessing).

All these intelligent agents will be sent to and evaluated by the MLVM, which is a multi-threading agent server framework. The whole MLVM architecture is designed as a collection of service modules (such as the creation module, the database module, etc.), and centred by a Prolog Virtual Machine (the engine).

Agents will be executed by the engine until a builtin predicate is hit. Then the engine will transfer the agent to a service module for proper operations. After finishing the corresponding operations, it will be returned back to the engine for continuous executions.

From the logic point of view, the implementation of Database Interface module is straightforward. But it is not the case on the technical standpoint. Due to the heavy-workload nature of database operations, they may cause serious delays or even permanent blocks on the system threads. Furthermore, temporary memories need to be allocated during certain database operation. In order to solve these problems, we discuss the following solutions.

3 CONNECTION MANAGEMENT

3.1 Multi-threaded Design

Instead of directly invoking database operations within system threads, the Multi-threading Database Connection management creates another bunch of Database Module threads for carrying database jobs. A proper module thread will be picked up from the Database Thread pool for the current operation under certain thread assignment strategies. Actually it is a classic "multi-provider/multi-consumer" problem and many existing algorithms can be adopted.

By introducing this kind of Database Module Threads, it is possible to unload those resource consuming database operations from system threads and avoid system threads to be delayed or blocked.

3.2 Different Connections Levels

Clearly all the remote database operations require Physical Database Connection (PDC) provided by the remote DBMS. In the common sense, both establishing and maintaining these PDCs are expensive. Three different levels of PDC assignment are discussed and compared: Predicate Level, Agent Level and Module Level.

On the predicate level, PDCs are distinguished by each database connection predicate. Whenever a "*db_connection*" predicate is triggered, a unique PDC will be established and kept active until the "*db_disconnection*" predicate de-actives it or the current agent moves out or terminates.

Obviously if large amount of agents require the database operations at the same time, most of network and memory resources will be occupied by those operations.

On the agent level PDCs are associated with agents. A unique PDC will be established and activated for each agent when it requires database connection at the first time. All the remaining database operations within this agent program will share the same PDC. An agent level PDC terminates only when the current agent moves out or terminates. Therefore the total number of concurrent active PDCs is limited and less resource is occupied than that in the predicate level. However the problem still exists if many agents engaged in the database operations concurrently. Furthermore the PDC resource seems not evenly distributed among all database jobs.

A unique PDC Pool is constructed on the module level. Whenever a Database Module Thread receives a database operation requirement from an agent, it will bind the current operation by a selected PDC from the PDC Pool and return it back after finishing the current operation. If there is no available PDC in the pool, current Database Module Thread will be blocked until some PDCs are returned by other database operations.

Because the total number of all active database connections is pre-defined, network resources charged by database connections can be anticipated. Due to this reason we believe that the module level PDC assignment is more suitable for the Mobile Agent system.

4 RESULT MEMORY POOL

For those database tuple retrieval operations ("*db_tuple*"), temporary caches need to be built for holding the searching results and these caches can be organized locally, remotely or both locally and remotely.

4.1 Local Result Memory Pool

The Local Result Memory Pool design means that the whole database searching result set will be downloaded onto the local Mobile Agent server, and the tuple retrieval operation can consume the searching the results entirely from the local server.

Because of the local storage of searching results, random cursor movement can be supported. This design is suitable for unstable network and offers reliable database tuple retrieval service. But more memory resources will be in charged for holding these temporary results.

4.2 Remote Result Memory Pool

Instead of loading the whole result set onto the local server, the job of maintaining search results can be taken by the remote DBMS. By referring to the position of a remote database cursor, current record can be located and fetched by database tuple retrieval operations through the established network connection.

It is obvious that the memory resources required by tuple retrieval operations can be decreased dramatically in this way. But the network connection between the Mobile Agent server and the remote database must keep active during those operations, which makes it suitable for reliable network environments. Because most of the DBMS systems do not support random cursor movement, agent program can only consume the result set in the consecutive order.

4.3 Combined Result Memory Pool

Different from Remote Result Memory Pool design, the result records can be stored in a local cache before being consumed, and a local cursor can be defined to point to the last record that has been transferred from remote database if all the records are consumed in consecutive order.

Under this combined design, the local cache grows gradually; therefore local memory consuming will be much less than that in Local Result Memory Pool design. The damage caused by network failure will be much less than that in the Remote Result Memory Pool design, because at least part of the result set has been locally cached.

5 MEMORY RELEASE

Two kinds of memory release strategies are introduced and may co-exist in the implemented Database Interface: Manual Memory Release and Automatic Memory Release.

The recycling of temporarily charged system resources can be manually triggered by invoking the specific predicate: "*db_disconnect*" in the context of agent program. For each Physical Database Connection used by the agent, it will be disconnected and deleted for the predicate and agent levels of PDC assignment strategies, or will be marked free and returned back to the PDC pool for the module level PDC assignment strategy. Considering those temporary charged memory blocks, they will simply freed by the system. In this way all the resources charged by the database operations can be released without any problems.

It is possible that these "*db_disconnect*" predicates may not be reached because of the recursive complexity or because that the agent developers simply forget to invoke these predicates at all. In these cases, another compromising method, known as Automatic Memory Release, is developed. A special memory releasing function handler can be inserted into the Agent Out module. Whenever an agent decides to terminate or move out, this function handler will be trigged by this module for searching and releasing those temporarily charged resources.

6 CONCLUSIONS

In this paper, a Multi-threaded Database Connection Management design is proposed, in which the Database Module Threads are developed for handling heavy-duty database jobs. Three levels of

Physical Database Connection assignment strategies are introduced and their characteristics are compared. For tuple retrieval operations, three different designs of the result memory pool are described and their performances are analysed. Furthermore the temporary resources charged during the database operations can be released both manually and automatically to prevent some serious problems. In the future, we will fully test these designs on the existing IMAGO system.

REFERENCES

Wielemaker, J. 2002, 'SWI-Prolog ODBC Interface', University of Amsterdam, The Netherlands.

Li, X. 2001a, 'IMAGO: A Prolog – based System for Intelligent Mobile Agents', *MATA'01*, Springer Verlag Lectures Notes in Computer Science, pp. 21-30.

Li, X. 2001b, 'An Alternative Framework for Intelligent Mobile Agents', *In Proceedings of IC-AI*, pp. 29-35.

Liang, H. & Li, X. 2003, 'IMAGO Prolog and Its Compilation', *In Proceedings of CCECE*.

Bueno, F., Cabeza, D., Carro, M., Hermenegildo, M., López, P. & Puebla, G. 2000, 'The Ciao Prolog System', Technical Report CLIP 3/97.1, School of Computer Science, Technical University of Madrid.

Applied Logic Systems Inc. 1999, *ALS Prolog ODBC Interface*, Cambridge, Ma, USA.

Draxler, C. 1992, 'Accessing Relational and Higher Databases Through Database Set Predicates in Logic Programming Languages', PhD thesis, University of Zurich.

Sheth, A.P. & O'Hare, A.B. 1991, 'The Architecture of BrAID: A System for Bridging AI/DB Systems', *In Proceedings of the Seventh International Conference on Data Engineering*, pp.570-581.

Nurcan, S. & Kouloumdjian, J. 1991, 'An Advanced Knowledge Base Management System Based on the Integration of Logic Programming and Relational Databases', *In Proceedings of IEEE*, pp.740-744.

Mckay, D.P., Finin, T.W. & O'Hare, A. 1990, 'The Intelligent Database Interface: Integrating AI and Database Systems', *In Proceedings of the 8th National Conference on Artificial Intelligence*, pp.677-684.

Nurcan, S., Li, L. & Kouloumdjian, J. 1990, 'Integrating Database Technology and Logic Programming Paradigm', *In : IAE/AIE 90, The Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, vol.1, pp.341-349.

Gozzi, F., Lugli, M. & Ceri, S. 1990, 'An Overview of PRIMO: A Portable Interface Between Prolog and Relational Databases', *Information Systems*, vol.15, no.5, pp.543-553.

Danielsson, M. & Barklund, J. 1990, 'Persistent Data Storage for Prolog', *In Proceedings of DEXA 90*.

Vieille, L., Bayer, P., Kuchenhoff, V. & Lefebvre, A. 1990, 'EKS-V1, A Short Overview', *AAAI 90 Workshop on Knowledge Base Management Systems*, Boston, USA.

Ceri, S., Gottlob, G. & Wiederhold, G. 1989, 'Efficient Database Access from Prolog', *IEEE Transactions on Software Engineering*, vol.15, no.2, pp.153-164.

Bocca, J., Dahmen, M. & Macartney, G. 1989, 'KB-Prolog User Guide', Technical Report, ECRC Munich.

Cuppens, F. & Demolombe, R. 1988, 'A Prolog-relational DBMS Interface Using Delayed Evaluation', *In Proceedings of the 3rd International Conference on Data and Knowledge Bases*, pp.135-148.

Nussbaum, M. 1988, 'Delayed Evaluation in Logic Programming: an Inference Mechanism for Large Knowledge Bases', *Reunión de interesados en el área de Sistemas Expertos de Suiza*.

Quintus Computer Systems Inc. 1987, *Quintus Prolog Database Interface Manual*, Mountain View, California.

Bocca, J.B. 1986, 'On the Evaluation Strategy of Educe', *In Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, pp. 368 – 378.

Jarke, M., Clifford, J. & Vasiliou, Y. 1984, 'An Optimizing Prolog Front-End to a Relational Query System', *ACM SIGMOD '84 Conference*, pp.296-306.

Chang, C.L. & Walker, A. 1984, 'PROSQL: A Prolog Programming Interface with SQL/DS', *Expert Database Workshop*, pp.233-246.