병렬 연역 데이타베이스에서 확장된 평가 알고리즘

조 우 현† · 김 항 준††

요 약

연역 데이타베이스가 병렬 컴퓨터 구조에 분할 적재될 때, 내포 술어에 대한 갱신이 결정적일 필요가 있으며, 이 결과를 이용한 내포 술어의 병렬 평가 알고리즘이 요구된다. 본 논문에서는 병렬 연역 데이타베이스의 내포 술어에 대한 삽입과 삭제가 결정적인 방법을 제안하고 병렬 연역 데이타베이스를 위한 병렬 컴퓨터 구조에서 갱신 방법이 고려된 확장된 병렬 평가 알고리즘을 제안한다. 연역 데이타베이스는 외연적 데이타베이스 즉 사실들의 집합과, 내포적 데이타베이스 즉 규칙들의 집합으로 구성된다. 이 집합들을 여러개의 처리기에 분산 적재하였을 때, 각각의 처리기에서 갱신 방법과 그 결과를 이용한 병렬 평가방법을 연구한다. 각각의 처리기는 자신의 지역 기억장치를 가지며 연결망을 통하여 서로 메세지를 교환함으로써 통신한다.

An Extended Evaluation Algorithm in Parallel Deductive Database

Woo-Hyun Cho† · Hang-Joon Kim††

ABSTRACT

The deterministic update method of intensional predicates in a parallel deductive database that deductive database is distributed in a parallel computer architecture is needed. Using updated data from the deterministic update method, a strategy for parallel evaluation of intensional predicates is required. The paper is concerned with an approach to updating parallel deductive database in which every insertion or deletion can be performed in a deterministic way, and an extended parallel semi-naive evaluation algorithm in a parallel computer architecture. After presenting an approach to updating intensional predicates and strategy for parallel evaluation, its implmentation is discussed. A parallel deductive database consists of the set of facts being the extensional database and the set of rules being the intensional database. We assume that these sets are distributed in each processor, reserch how to update intensional predicates and evaluate using the update method. The parallel architecture for the deductive database consists of a set of processors and a message passing network to interconnect these processors.

1. Introduction

A parallel deductive database is a deductive database which is distributed to the parallel computer architecture with any partition procedure. A parallel deductive database consists of a set of facts and a set of rules; the set of facts being the extensional database (EDB) and the set of rules being the intensional database(IDB)[14]. A parallel computer architecture for the parallel deductive database assumes more than one

† 정 회 원:부경대학교 전자계산학과 부교수
†† 종신회원:경북대학교 컴퓨터공학과 교수
논문접수:1996년 4월 26일, 심사완료:1996년 8월 19일

processor with each having its own memory. A processing element(PE) consists of a processor, a local main-memory and an optional secondary storage device.

Parallel computer architectures are classified according to the organization of their processors and memory units, the style of computation, and so on. In a shared everything architecture, main-memory and secondary storage devices are shared across the processors. In a shared disk approach, disks are shared across the processors. In this paper, we consider only a shared-nothing architecture. This architecture consists of a set of PEs and a message passing network to interconnect these PEs. Hence it could be either a switched or a networked system. For parallel deductive database applications, each processing element owns a portion of the deductive database, in other words, the deductive database is partitioned amongst the multiple processing elements.

The problem of updating intensional predicates in the deductive database has attracted considerable attention in recent years. This problem consists in translating an insertion or deletion of a fact over an intensional predicate to possibly more than one insertion or deletion of a fact over the extensional predicates. For a given insertion or deletion, more than one such translation is refered to in the literature as non-determinism of the translation. We explain this problem using a simple example. Consider the following deductive database.

Extensional database(facts):

$r_1(a, 1)$ $r_2(1, b)$ $r_3(a, b)$ $r_3(a, c)$

Intensional database(rules):

$r_4(X, Y) \leftarrow r_3(X, Y)$
$r_4(X, Y) \leftarrow r_1(X, Z), r_2(Z, Y)$
$r_5(X, Y, Z) \leftarrow r_1(X, Y), r_2(Y, Z)$

As a first example of update, suppose that a user wants to insert the fact $r_4(a, d)$ without giving further information as to the extensional database. We have two ways to translate the update; either insertion of the fact $r_3(a, d)$ or insertion of the facts $r_1(a, X)$ and $r_2(X, d)$. Clearly, in either case, the resulting database implies the fact $r_4(a, d)$ that was to be inserted. However, we have non-determinism of the translation, possibly at more than one level. If we choose to insert fact $r_3(a, d)$, then no further choice is needed. If, however, we choose to insert the facts $r_1(a, X)$ and $r_2(X, d)$, then we must also choose a value of X, so that we can infer $r_4(a, d)$ from $r_1(a, X)$ and $r_2(X, d)$. In case that user dose not want such updates, the insertion of $r_4(a, d)$ cannot be performed. So, clearly, the only way to perform the insertion is to simply store the fact $r_4(a, d)$ in the processing element for the intensional database.

As a second example of update, suppose that the user wants to delete the fact $r_5(a, 1, b)$. This fact is not present in the extensional database but can be infered using the third rule. Again, there are two ways to translate this deletion: either deletion of the fact $r_1(a, 1)$ or deletion of the fact $r_2(1, b)$. Clearly, in either case, the resulting database does not imply the fact $r_5(a, 1, b)$ and thus the intended deletion is performed. The choice as to which of the two deletions should be executed is also left to the system. However, it might be the case that a user wishes to delete the fact $r_5(a, 1, b)$ without performing further deletions. Then the only way to perform the deletion is to store the fact $r_5(a, 1, b)$ as being a deleted fact in the processing element.

In a shared nothing architecture for a deductive database, each processing element contains extensional database or intensional database, inserted facts and deleted facts. An evaluation for each processing element can be regarded as a sequence of rule-instantiations using inserted facts and deleted facts. In each rule-instantiation, the variables in the rule are replaced by constants from facts. The purpose of the parallel evaluation is to partition the instantiations among multiple processors. For a given facts and

rules, the evaluation time of a parallel evaluation algorithm depends on two factors. One is the maximum workload of a processor, and the second is the communication overhead.

The determinstic update method of intensional predicates in the deductive database was discussed and it's correctness was described in ref[1, 10, 11]. The parallel semi-naive evaluation(PSNE) algorithm of intensional predicates in the parallel deductive database was proposed in ref[17]. The PSNE algorithm was based on the work in ref[14]. However, the issues discussed in this paper had not been addressed previously. In this paper, we introduce the deterministic update method of intensional predicates in the parallel deductive database, and extend the PSNE algorithm to use updated data.

2. Basic Definitions and Notations

In this section, we define the basic terminology. A literal is a predicate symbol followed by a list of arguments. An atom is a literal with a constant or a variable in each argument position. A constant is any natural number or any string with one or more lower alphabets. The other arguments of an atom are variables such that X, Y, and Z. If an atom has a constant in each argument position, then it is a fact. A rule consists of an atom designated as a head, and a conjunction of one or more atoms, designated as the body. We assume that every variable in the head of a rule appears in the body of the rule and that all rules are rectified. The instantiation of the rule is the replacement of each variable in the rule by a constant. A deductive database is a finite set of facts and rules whose predicate symbols are divided into two disjoint subsets: the extensional predicates i.e. the extensional database(EDB), and the intensional predicates i.e. the intensional database(IDB). The extensional predicates do not appear in any head of a rule. An extensional relation is a collection of facts with same extensional predicate symbol, and an intensional relation is a collection of facts with same intensional predicate symbol, made by instantiation of rules.

For a rule r, an arithmetic predicate of the form h (X, ..., Z), where X, ..., Z are distinct variables, each of which appears in r, is called a restricting predicate. A restricting version r′ of a rule r is obtained by appending to the body of r a restricting predicate. A restricting version for all intensional predicates is a collection of rules that is obtained by replacing each rule by a restricted version of it. We assume that only restricted versions have arithmetic predicates.

Let Γ_0 be an initial deductive database with facts and rules, that we denote $\{r_1, r_2, ..., r_k\}$. Γ_0 is partitioned among the multiple processing elements for parallel evaluation. Assume that there are m processing elements, $\{PE_1, PE_2, ..., PE_m\}$. A partitioned deductive database which a processing element contains, Γ_{PEi} is a 4-tuple $\Gamma_{PEi} = (IF_i, DF_i, F_i, R_i)$, where IF_i is a finite set of inserted facts for intensional predicates, DF_i is a finite set of deleted facts for intensional predicates, F_i is a finite set of facts for extensional predicates and R_i is a finite set of rules. Initially, Γ_{PEi} is $(\{\}, \{\}, F_i, R_i)$. The set $\{\Gamma_{PE1}, \Gamma_{PE2}, ..., \Gamma_{PEm}\}$ is partitioned from Γ_0 for each processing element.

Let Q be some predicate in an intensional database. We define that the output for Q is the set of Q-facts that have a derivation tree in the deductive database given EDB and IDB. A derivation tree for a fact, a, is a finite tree with nodes labeled by facts. The root is a, the leaves are facts in EDB or IF_i, and for each internal node, b, with children $b_1, b_2, ..., b_n$, there is an instantiation of a rule of IDB that has b as the head and $b_1, b_2, ..., b_n$, as the body. If a rule is a restricted version, then the instantiation must satisfy the restricting predicate. The output of given IDB is the union of the output for all the intensional predicates. The set of EDB and output of IDB is called the global view of the deductive database Γ. The local view for Γ_{PEi} is the union of EDB and output of IDB which PE_i contains. In other words, the global view of the deductive database Γ is the union of all local

views and the meaning of Γ. This is the proof-theoretic view.

3. Updating Intensional Predicates

Our approach to updating intensional predicates relies on the following three assumptions. First, the user can insert or delete any fact that he wishes, that is, the user can insert or delete facts over any extensional or intensional predicates. However, the user can not change the set of rules. Second, the system stores both inserted and deleted facts. That is, deleted facts over any intensional predicate are not removed from the deductive database, but are stored in much the same way as inserted facts are. Third, when user wishes facts over any extensional or intensional predicates to be inserted or deleted, the system can insert or delete them in the processing element which contains corresponding predicates.

Deleted facts that are stored in the deductive database play an active role and have a direct influence on database consistency and evaluation. A deductive database Γ contains no fact which is both inserted and deleted. Moreover, given a fact f, we say that Γ derives f if either f is inserted or f is not deleted and f can be derived from inserted facts and negations of deleted facts, using the rules.

Let $\Gamma_{PEi}=(IF_i, DF_i, F_i, R_i)$ be a partitioned deductive database for a processing element PE_i, and let f be a fact.

The insertion of f in Γ_{PEi}, denoted by $ins(\Gamma_{PEi}, f)=(IF_i', DF_i', F_i', R_i)$, is a database defined by the following steps:

step 1: if f is an extensional predicate then $F_i'=F_i \cup \{f\}$

step 2: otherwise, [if $f \in IF_i$ then $IF_i'=IF_i-\{f\}$;

step 3: if $f \in DF_i$ then $DF_i'=DF_i-\{f\}$]

The deletion of f in Γ_{PEi}, denoted by $del(\Gamma_{PEi}, f)=(IF_i', DF_i', F_i', R_i)$, is a database defined by the following steps:

step 1: if f is an extensional predicate then $F_i'=F_i-\{f\}$

step 2: otherwise, [if $f \in IF_i$ then $IF_i'=IF_i-\{f\}$;

step 3: $DF_i'=DF_i \cup \{f\}$].

In other words, in oder to insert a fact f over an intensional predicate we just add f to the inserted facts and remove f from deleted facts; in oder to delete a fact f over an intensional predicate we just remove f from the inserted facts and add f to the deleted facts. There are two important properties of updates. First, every insertion or deletion is possible with the deterministic method by system because of inserting or deleting a fact f over the intensional predicate. Indeed, a consistent database remains consistent after every insertion or deletion. Second, the insertion or deletion of a fact requires simply the addition and removal of a single fact. An inference of facts is not required during updating. So, this approach to updating leads to a deterministic translation process.

4. Extended Parallel Semi-naive Evaluation Algorithm

Let Γ_0 be an initial deductive database with facts and rules, that we denote $\{r_1, r_2, ..., r_k\}$, and $\{PE_1, PE_2, ..., PE_m\}$ be a set of $m \geq 1$ processing elements. The set $\{\Gamma_{PE1}, \Gamma_{PE2}, ..., \Gamma_{PEm}\}$ to be partitioned from Γ_0 for each PE, is called a parallelization strategy for Γ_0. Assume that the set of rules in the Γ_{PEi}, denoted by R_i, is $\{r_{ij} | 1 \leq j \leq n_i\}$ and I_{ij} is an intensional relation corresponding to the predicate r_{ij}. The set of PEs $\{PE_1, PE_2, ..., PE_m\}$ cooperate in evaluating Γ in parallel. Processing element PE_i performs the instantiations of the rules in the Γ_{PEi}. The instantiations of PE_i can be performed by using any single-processor evaluation method on Γ_{PEi}. However, the method has to be adjusted, to account for additions to the local view made by other PEs, not just PE_i, and to consider update facts. The parallel algorithm ends when none of the PEs can perform an instantiation of a rule in its Γ_{PEi}, such that a new fact is added to the local view.

Now assume that there is a local memory for each PE to store the local view. Assume further that the facts are either replicated or transmitted to all the PEs. The message-passing or shared-nothing variation of the evaluation strategy using updated facts is as follows. Each processing element, PE_i, starts with the local view consisting of the facts to Γ_{PEi}, performs the instantiations of rules to Γ_{PEi}, and modifies the local view using IF_i and DF_i. PE_i transmits to each other processing element, PE_j, the set of tuples that PE_i computes. This set may be less than the whole set of tuples computed by PE_i. PE_i also receives from each other PE the set of tuples the latter computed. The communication among the PEs is totally asynchronous during computation. The algorithm performed by PE_i is a variation of the procedure below. The procedure is executed iteratively, until the termination condition is satisfied.

[Evaluation-Procedure]

1) Add to the local view new tuples obtained by instantiations of rules in the Γ_{PEi}
2) Remove deleted facts from the local view, and add inserted facts to it.
3) Transmit to the other PEs some of the new tuples computed.
4) Receive from the other PEs some of new tuples and add them to the local view.

We consider algorithms based on the semi-naive evaluation of rules. Communication among the PEs is by message passing. Extended-PSNE algorithm is executed by some processing element, PE_i. We use Ullman's notation for the seminaive evaluation algorithm. EVAL-INCR(r_{ij}, E_1, ..., E_t, I_{11}, ..., Imn_n, ΔQ_1, ..., ΔQ_n) is a function that computes the tuples of I_{ij} that can be obtained by the instantiations of rule r_{ij}, given extensional relations E_1, E_2, ..., E_t, intensional relations I_{11}, I_{12}, ..., Imn_m, and incremental relations ΔQ_1, ΔQ_2, ..., ΔQ_n. EVAL(r_{ij}, E_1, ..., E_t, 0, ..., 0) is similar[14].

[Extended-PSNE Algorithm for PE_i]

1 for j:= 1 to n_i do begin
2 $\Delta I_{ij} := EVAL(r_{ij}, E_1, ..., E_t, 0, ..., 0)$;
3 $S_{ins} := \{f \mid f \in IF_i$, f is an inserted fact for the predicate $r_{ij}\}$;
4 $S_{del} := \{f \mid f \in DF_i$, f is a deleted fact for the predicate $r_{ij}\}$;
5 $I_{ij} := \Delta I_{ij} + S_{ins} - S_{del}$;
6 end;
7 send from the ΔI_{ij}'s the subset T_{ip}, to each $PE_p(1 \le p \le m)$;
8 repeat
9 for j:= 1 to n_i do $\Delta Q_j := \Delta I_{ij}$;
10 add each tuple received from the other PEs during last iteration or waiting, into corresponding intensional relation;
11 for j:= 1 to n_i do begin
12 $\Delta I_{ij} := EVAL\text{-}INCR(r_{ij}, E_1, ..., E_t, I_{11}, ..., Imn_m, \Delta Q_1, ..., \Delta Q_n)$;
13 $S_{del} := \{f \mid f \in DF_i$, f is a deleted fact for the predicate $r_{ij}\}$;
14 $\Delta I_{ij} := \Delta I_{ij} - I_{ij} - S_{del}$;
15 end;
16 send from the ΔI_{ij}'s the subset T_{ip}, to each $PE_p(1 \le p \le m)$;
17 for j:= 1 to n_i do $I_{ij} := I_{ij} \cup \Delta I_{ij}$;
18 until $\Delta I_{ij} = \phi$ for all j's;
19 wait until PE_i detects termination or receives some tuples from other PEs;
if termination detection, then output $\{I_{i1}, I_{i2}, ..., I_{in_i}\}$ and exit;
if tuples received, then goto step8.

[End of Algorithm]

In the Extended-PSNE algorithm, one iteration (step1-6) is performed for initial facts of intensional predicates using inserted facts or deleted facts. A subsets of the newly computed tuples in step1-6 are transmitted to all the other processors in step7. The other iteration(step8-18) computes new tuples for intensional predicates with all the tuples received from other processors during the last iteration are added to the local database and with the deleted facts in the local database. Results of the step12 and S_{del} can be incon-

sistent. To avoid this inconsistency, the step14 operation is needed.

Now we consider the termination condition. In our discussion, a parallel computation consists of a set of PEs, which communicate with one another via message passing. Termination should occur when each PE has reached a fixed-point and there are no message in transit. For detecting termination, one has only to select an algorithm from the many published in existing literature. A PE is either idle or active. An active PE may become idle at any time. Only active PEs may send messages to others; and, an idle PEs can only be reactivated by receiving a message. The algorithms provided in the literature superimpose a termination detection algorithm on the computation[4, 6, 9, 13]. In our terminology, a PE is idle if it reaches a temporary fixed-point, otherwise it is active. A PE reaches a temporary fixed-point if by instantiating the rules, new tuples that do not exist in the local view, cannot be generated. The computation is said to be terminated if and only if all the PEs are idle and there is no message in transit.

5. Discussion

In this paper, we introduced the extended PSNE algorithm for evaluating intensional predicates in the parallel deductive database using updated facts. The deductive database is partitioned for each processing element in the shared-nothing architecture. A deterministic approach to updating intensional predicates in each processing element was discussed. In this approach, however, the storage of deleted or inserted facts introduces a significant overhead. Each processing element performs the instantiations in a partition member using some evaluation method, adds the newly generated tuples to a local view, and sends them to other processing elements. The transmission set T_{ij} that PE_i sends to PE_j consists of the intersection of the set of Q-facts computed by PE_i and the set of Q-facts used by PE_j. We intend to continue the research of an augmentation of parallel evaluation with some interesting parallelization ideas that introduced in some literatures[2, 3, 5, 7, 15].

References

[1] Atzeni, P. and Torlone, R., "Updating Intensional Predicates in Datalog," Data and Knowledge Engineering, 8, 1992, pp.1-17.

[2] Chassin, J. and Codognet, P., "Parallel Logic Programming Systems," ACM Computing Survey, Vol.26, No.3, September 1994, pp.295-336.

[3] Das, S.K., "Deductive Database and Logic Programming," Addison Wesley, 1992.

[4] Eriksen, O., "A termination Detection Protocol and Its Formal Verification," Journal of Parallel and Distributed Computing 5, 1988, pp.82-91.

[5] Ganguly, S., Silberschatz, A. and Tsur, S. "A framework for the parallel processing of Datalog queries," Proceedings of the ACM SIGMOD International Conference on Management of Data, May 1990, pp.143-152.

[6] Huang, S.T., "Detecting Termination of Distributed Computations by External Agents," The 9th International Conference on Distributed Computing Systems, June 1989, pp.79-84.

[7] Hulin, G., "Parallel processing of recursive queries in distributed architecture," Proceedings of the 15th International Conference on Very Large Data Bases, August 1989, pp.87-96.

[8] Kowalski, R.A., "Logic for Problem Solving," Elsevier Science, New York, 1979.

[9] Lai, T.H, "Termination Detection for Dynamically Distributed Systems with Non-first-in-first-out Communication," Journal of Parallel and Distributed Computing 3, 1986, pp.577-599.

[10] Laurent, D., Phan Luong, V. and Spyratos, N., "Deleted Tuples are Useful when Updating through Universal Scheme Interfaces," 8th IEEE ICDC, Pheonix, 1992.

[11] Laurent, D., Phan Luong, V. and Spyratos, N.,

"Updating Intensional Predicates in Deductive Database," The 9th International Conference on Data Engineering, April 1993, pp.14-21.

[12] Lloyd, J.W., "Foundations of Logic Programming," Springer-Verlag, Berlin, 1987.

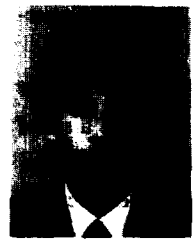
[13] Mattern, F. "Algorithm for distributed termination detection," Distributed Computing 2, 1987, pp.161-175.

[14] Ullman, J.D., "Principle of Database and Knowledge-Base Systems," Vol 1, Computer Science Press, 1988.

[15] Valduriez, P. and Khoshafian, S., "Parallel evaluation of the transitive closure of a database relation," Int. J. Parallel Programming, vol.17, no. 1, Feb. 1988.

[16] Van Gelder, A., "A message passing framework for logic query evaluation," Proceedings of the ACM SIGMOD International Conference on Management of Data, May, 1986, pp.155-165.

[17] Wolfson, O. and Ozeri, A., "Parallel and Distributed Processing of Rules by Data-Reduction," IEEE Transaction on Knowledge and Data Engineering, Vol 5, No.3, June 1993, pp.523-530.



조 우 현

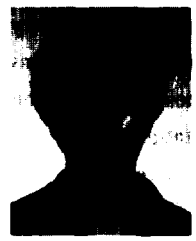
1985년 경북대학교 전자공학과 전산공학전공 졸업(공학사)

1988년 경북대학교 대학원 전자공학과 전산공학전공(공학석사)

1988년~1989년 한국전자통신연구소 연구원

1989년~현재 부경대학교 전자계산학과 부교수

관심분야: 데이타베이스, 인공지능, 병렬처리



김 항 준

1977년 서울대학교 전기공학과 졸업(공학사)

1979년 한국과학기술원 전기 및 전자공학과(공학석사)

1980년~현재 경북대학교 컴퓨터공학과 교수

관심분야: 인공지능, 문자인식, 음성인식, 병렬처리, 컴퓨터구조