Concurrency Control Optimizations in a Prolog Database

Soumitra Sengupta and Arthur J. Bernstein

Received March 1990; revised October 1990

The degree of concurrency allowed by a concurrency control scheme depends on the knowledge it has about the database and the transactions. This paper analyzes the syntactic information available in a Prolog database to improve concurrency. An optimistic concurrency control scheme is chosen since it is a natural choice for Prolog databases. The conflict criteria among transactions in a Prolog environment is translated into conditions on the query search tree. This aids in identifying concurrency related information from the depth-first search execution strategy. A spectrum of validation algorithms develops naturally. The algorithms are presented in stages such that progressively more information is considered to construct the read and the write sets (required to check conflicts) and better concurrency is achieved. An orthogonal contribution of the information analysis is a new query evaluation optimization. The price for better concurrency is paid in terms of increased storage and computation costs. The optimality of the algorithms with respect to the available information and derived concurrency is discussed qualitatively.

KEY WORDS: Transactions; Prolog databases; optimistic concurrency control; optimizations.

1. INTRODUCTION

The recent trend in database management systems is to introduce deductive capabilities into relational databases. Use of the logic language Prolog is becoming an increasingly popular choice because of the natural corre-
spondence between Prolog predicates and expressions of relational algebra and between Prolog facts and tuples of relations. Prolog fulfills the roles of the database definition and query languages, and since it is a language based on first-order logic, it supports natural but sophisticated deduction procedures as part of the language. Much current research is directed towards defining appropriate interfaces to bring relational databases and Prolog systems closer to each other.\(^{(1-3)}\)

A database normally exists in a dynamic environment where many users perform activities concurrently. The integrity of the database in a multi-user environment is maintained by imposing a transactional mechanism with appropriate concurrency control.\(^{(4-6)}\) Therefore, a Prolog database implementation must have embedded in itself a transaction processing facility with requisite properties of atomicity and recoverability.\(^{(7)}\)

Concurrency control algorithms can be broadly classified as pessimistic and optimistic. The pessimistic algorithms (conservative 2-phase locking, \(^{(6,8)}\) timestamp ordering\(^{(9)}\)) check for conflicts at each access to a data item, and if detected, either delay or rollback the transaction. The optimistic schemes (also known as certification and aggressive schedulers) assume that conflicts are rare and that it is, therefore, wasteful to synchronize at each access. Instead, the transactions are allowed to access items freely, and at the commit point, validation checks are made to determine if a conflict has occurred. Also, to avoid temporary inconsistencies in the database, the updates of a transaction are deferred until it has been validated successfully.\(^{(6,10,11)}\)

The amount of concurrency allowed by a concurrency control scheme depends on the knowledge available to it about the database and the transactions. Usually, such information is also classified in two categories, namely, syntactic and semantic. Syntactic information relates to knowledge of properties such as granularity of the data items and access modes. The query evaluation strategy of transactions in terms of the data item access patterns also provides syntactic information. On the other hand, semantic information describes the items themselves and the effects of applicable operations. It is possible to improve concurrency in the system using this additional information.

In this paper, we analyze various levels of syntactic information that may be available in a Prolog database system. The achievable degree of concurrency is dependent on the level. We consider a query-dominant environment where conflicts are naturally rare. Also, we choose to implement a Prolog system that uses logical operators \texttt{assume} and \texttt{desume} for updating the database instead of \texttt{assert} and \texttt{retract}.\(^{(12,13)}\) A transaction, after writing an item using \texttt{assume} or \texttt{desume}, may have to retract the write action due to failure backtracking at a later time; which is the intended