Partitioned Data Objects in Distributed Databases

G. ALONSO AND A. EL ABBADI

Department of Computer Science, University of California, Santa Barbara, CA 93106

Received September 2, 1993; Revised July 8, 1994

Recommended by: Hector Garcia-Molina

Abstract. In many distributed databases "locality of reference" is crucial to achieve acceptable performance. However, the purpose of data distribution is to spread the data among several remote sites. One way to solve this contradiction is to use partitioned data techniques. Instead of accessing the entire data, a site works on a fraction that is made locally available, thereby increasing the site's autonomy. We present a theory of partitioned data that formalizes the concept and establishes the basis to develop a correctness criterion and a concurrency control protocol for partitioned databases. Set-serializability is proposed as a correctness criterion and we suggest an implementation that integrates partitioned and non-partitioned data. To complete this study, the policies required in a real implementation are also analyzed.

Keywords: Partitioned data, distributed databases, concurrency control, recovery

1. Introduction

In a distributed database, overall performance is greatly influenced by the amount of communication required to execute an operation. Faster response time and higher throughput can be achieved if all the data accessed by a transaction is available at the site where the transaction is initiated. This is rarely the case in a conventional distributed database. However, there are applications in which it is possible to partition some data (commonly aggregated fields) among all the sites so that transactions are executed on the fraction of the data that is locally available. Since sites require less communication with other sites, partitioned data techniques increase the autonomy of each site and the overall throughput.

In spite of the extensive work done on the subject there is no formal theory of partitioned data. As a result, it is difficult to compare the different approaches and to evaluate their advantages. Moreover, most of the proposed protocols apply only to partitioned data, leaving the integration of partitioned and non-partitioned data as an open question. The lack of a formal model also makes it difficult to measure the implications of partitioned data in terms of resilience to failures and recovery.

In this paper we develop such a formal theory, and propose a new concurrency control protocol based on the notion of set-serializability that expands serializability and integrates partitioned and non-partitioned data under a homogeneous concurrency control system. We also propose an implementation of this protocol, taking advantage of the formalisms developed in the first part of the paper. An analysis of partitioned data would be incomplete without discussing the actual policies involved in an implementation, we perform such a study both through a theoretical analysis and a simulation of the policies.

The paper is organized as follows. In Section 2 we sketch the general background of partitioned data techniques and related work. Section 3 introduces set-serializability as the
concurrency criterion for partitioned data. In Section 4 we discuss the implementation of a concurrency control protocol based on set-serializability. Section 5 contains a simulation analysis of the different policies that can be used in the implementation of the proposed protocol. Section 6 concludes the paper.

2. Background

In this section we start by motivating the problem of partitioned data and describe some of its distinctive features. We then describe previous work related to the definition of distributed assertions, partitioned data protocols and their application to fault tolerance, and data replication.

2.1. Motivation

To illustrate the intuition behind data partition techniques, consider an application where a data item represents the number of tickets to be sold for some event. In a conventional database these tickets reside in a central site. Requests for tickets originating at the central site can be satisfied locally. All other sites in the system must exchange a series of messages with the central site, which results in delays in the processing of transactions that request tickets. Instead of using this approach, a partitioned database divides the “tickets” data item among all the sites. Each site is allocated a fraction of the tickets and will use them to process transactions as long as it has enough tickets locally available. As a result, the communication overhead is avoided for most transactions.

This simple idea has proven elusive to formalize and implement. The inherent distributed nature of partitioned data makes it difficult to design a scheduler that guarantees correct executions both on partitioned and non-partitioned data. Moreover, the particular characteristics of these systems allow transactions to cooperate, in the sense that a group of transactions may become mutually dependent in their execution. These executions are correct but existing models tend to discard them as invalid for recovery purposes and because traditional correctness criteria such as serializability cannot deal with cyclic dependencies.

Partitioned data display a variety of characteristics that make them unlike any other data. Distributed in nature, a partitioned data item requires control not only of the concurrent execution of transactions but also of the semantics of those transactions. In conventional databases, by definition, a transaction, executed in its entirety and isolated from other transactions, leaves the database in a consistent state. Thus a database constraint affecting a single data item remains locally verifiable in a distributed database. In a partitioned database, such constraints become distributed assertions that must be maintained across all the system’s sites. In the ticket sale example, a non-overbooking constraint is easy to enforce in a centralized system. This is not the case when the tickets are partitioned, since enforcing such constraints requires agreement among all sites.

This added complexity to the design of the concurrency control is compensated for by the benefits of partitioned data. The increase in a site’s autonomy directly results in a higher transaction throughput and faster response time for all operations on partitioned data.