A Parallel Implementation Scheme of Relational Tables Based on Multidimensional Extendible Array

K. M. Azharul Hasan, University of Fukui, Japan
Tatsuo Tsuji, University of Fukui, Japan
Ken Higuchi, University of Fukui, Japan

ABSTRACT

In this article, an efficient parallel implementation scheme of relational tables is proposed and evaluated. The scheme implements a relational table by employing an extendible multidimensional array. Data allocation is a key performance factor for parallel database systems. This holds especially for data warehousing environments in which huge amounts of data have to be dealt with. In our scheme, an efficient data allocation technique is used, based on the notion of extendible array. The dynamic load balancing is conducted when load on each processor is not uniformly distributed in order to maximize processor utilization.

Keywords: extendible array; load balancing; multidimensional array; parallel database; parallel retrieval; relational table

INTRODUCTION

Database systems increasingly rely upon parallelism rather than upon a single monolithic processor in order to achieve high performance and large capacity (DeWitt & Gray, 1992). Much research has been done on the design, implementation, and performance analysis of parallel database systems (PDBS) (Boral et al., 1990; Dewitt et al., 1990; Rao, Zhang, Lohman, & Megiddo, 2002). High throughput and response time can be achieved by applying parallelism for complex queries. Each processor processes a portion of the database. The degree of parallelism is determined by the distribution of the data in the system. Load balance, therefore, is an important issue to the performance of parallel databases. When the data are not distributed uniformly among the processors, rebalancing the data load is necessary in order to ensure good system performance (Bouganim, Florescu, & Valduriez, 1996; Hua & Su, 1993; Lakshmi & Yu, 1998; Taniar & Leung, 2003).

In this article, a new parallel implementation scheme of relational tables is proposed and
evaluated. The scheme implements a relational table by employing a multidimensional array. Each dimension of the array corresponds to a column of the table, and each position of the array indicates the corresponding record. However, these kinds of multidimensional arrays suffer from some problems:

1. In general, the implemented arrays are very sparse.
2. The length of each dimension is statically predetermined; hence, dynamic extension of the length of the array is impossible if the size of the dimension overflows.

In order to solve problem 2, the concept of extendible array (Otoo & Merrett, 1983; Rosenberg, 1974) will be employed. An extendible array is extendible in any direction without any relocation of the data already stored. Moreover, we design a data structure that stores only the effective array elements existing as actual records of the relational table; this solves problem 1.

Data allocation is a key performance factor for parallel database systems. This holds especially for data warehousing environments in which huge amounts of data have to be dealt with (Stohr, Martens, & Rahm, 2000). In general, the PDBS handles relational tables; data allocation is based on horizontal fragmentation of tables produced typically in round-robin, hash, or range fragmentation (DeWitt & Gray, 1992). Round robin simply distributes rows in their insert order, while hash and range fragmentation are based on a partitioning function applied to the values of fragmentation attribute. Horizontal as well as vertical fragmentation is proposed to achieve maximum parallelism by Sacca and Wielderhold (1985).

In this article, a new data allocation technique is used. This allocation is based on the extension history of the underlying extendible array and the offset value (i.e., location) in the subarray. In our data allocation scheme, the candidate and non-candidate records for a query are separated based on the notion of subarray. In our previous work (Hasan, Kuroda, Azuma, Tsuji, & Higuchi, 2005), the History-Offset implementation of Relational Tables (HORT) and its superiority over conventional implementation of relational tables was presented. In this article, we have reorganized our HORT data structure to be implemented in a parallel environment. Our scheme will be called P-HORT (Parallel History-Offset implementation of Relational Tables). The proposed data allocation and processing model supports parallel I/O and parallel processing as well as load balancing for secondary storage and processors. Our data allocation method is fully general and can be applicable to all major multiprocessor database architectures (e.g., shared nothing, shared disk, and shared everything) (Bhide, 1988). It can be applied effectively not only to the implementation of relational tables for parallel databases but also to parallel data warehouse systems (Muto & Kitsuregawa, 1999; Stohr et al., 2000).

THE NOTION OF EXTENDIBLE ARRAY

An n dimensional extendible array A has a history counter h and three kinds of auxiliary tables for each extendible dimension \( i = 1, \ldots, n \) (see Figure 1). These tables are history table \( H_i \), address table \( L_i \), and coefficient table \( C_i \). The history tables memorize extension history \( h \). If the size of A is \([s_n, s_{n-1}, \ldots, s_1]\) and the extended dimension is \( i \), for an extension of A along dimension \( i \), contiguous memory area that forms an \( n-1 \) dimensional subarray S of size \([s_n, s_{n-1}, \ldots, s_{i+1}, s_{i-1}, \ldots, s_2, s_1]\) is dynamically allocated. Then, the current history counter value is incremented by one, and it is memorized on the history table \( H_i \); also, the first address of S is held on the address table \( L_i \). Since \( h \) increases monotonously, \( H_i \) is an ordered set of history values. As is well known, an element \( (i_1, \ldots, i_n) \) in an n dimensional conventional fixed size array of size \([s_n, s_{n-1}, \ldots, s_1]\) is allocated on memory using an addressing function like this:

\[
\begin{align*}
  f(i_n, i_{n-1}, i_{n-2}, \ldots, i_2, i_1) &= s_1s_2\cdots s_{n-1}i_n + s_1s_2\cdots s_{n-2}i_{n-1} \\
  &\quad + \ldots + s_1i_2 + i_1.
\end{align*}
\]
The IT Readiness for the Digital Universe
www.igi-global.com/chapter/the-it-readiness-for-the-digital-universe/150192?camid=4v1a