Parallel and Distributed Solutions for the Optimal Binary Search Tree Problem

Mitică Craus

"Gh.Asachi" Technical University of Iasi
Computer Engineering Department
6600 Iaşi, Romania
craus@cs.tuiasi.ro

Abstract. A parallel and a distributed implementation for a very important problem in the searching theory, the optimal binary search tree (BST) problem, is presented and analyzed. Implemented as a VLSI array, the algorithm for building the optimal BST uses $O(n^2)$ processors and has the parallel time complexity $O(n)$. A search is solved in $O(\log n)$ time. On a cluster of computers, the binary search tree is organized on two levels: the first level corresponds to the BST of searching intervals and the second level is the level of the BST for effective searching within an interval. A hybrid solution is also considered. The best variant depends on the hypothesis of the searching problem.

1 Distributed Searching Structures and Algorithms

Usually, a wide area distributed database has not a single index of all information. That is the case of a national or regional network of electronic libraries or a distributed internet searching engine.

Which is the most adequate data structure for a distributed index? There are a lot of searching data structure (optimal binary search trees, AVL trees, 2-3 trees, 2-3-4 trees, B-trees, red-black trees, splay trees, digital search trees, tries [1,5,6]) but are they enough for a distributed index? The answer seems to be negative.

The internal relationships of an index usually have the topology of a digraph and this data structure is not adequate for efficient search algorithms. Building spanning tree [3] in a dynamically manner on the digraph representing the key relations in the index may be a way to use the search algorithms for trees. On the other hand, a distributed index imposes the distribution of the digraph over the network.

Also, the search algorithms for distributed search data structures differ from that for locally search data structures.

In a distributed search engine, a single user query starts in one place and the result should be returned to that one place. Centralized control of the searching process will direct where to search next and determine when the search is completed. With a fully distributed search, there is no central control.
Both searching with centralized control and with no central control have advantages and disadvantages.

The main advantage of a centralized search is the control of the searching process but a centralized controller managing the entire process can become a bottleneck.

The advantages of a distributed search is that it offers greater processing power with each remote server performing part of the search, and avoids the bottleneck of a centralized controller managing the entire process. One problem with no central control is that often the same area is searched multiple times. Another problem of distributed searching is uniformity of evaluation. With several search engines each performing their searches using their own criterion for ranking the results, the synthesis of the various results is a difficult problem.

Which is the best solution for searching in a distributed index: centralized or distributed search? This is a problem. The answer seems to be the distributed search but the centralized search has its advantages that cannot be neglected.

Our paper is concerning with the problem of building a distributed search structure adequate to very fast centralized searches. This structure is a distributed binary search tree. It may be used for indexing a wide area distributed database. There are some researches in the indexing information retrieval area [4] but we have no information to be one referring to a distributed index based on binary search tree, organized in such a manner that the keys with the high probability of searching are reached faster then the others.

2 Optimal Binary Search Tree Problem

Let us consider \( A = (a_1, a_2, ..., a_n) \) a sequence of data items increasing sorted on their keys \((k_1, k_2, ..., k_n)\). With these items it follows to be built a binary search tree [1].

Each item \( a_i \) is searched with the probability \( p_i, i = 1, 2, ..., n \). Let us denote by \( q_i \) the probability to search the item \( x \) with the property that \( a_i < x < a_{i+1}, i = 0, ..., n \), where \( a_0 = -\infty, a_{n+1} = +\infty \). In these conditions \( \sum_{i=1}^{n} p_i \) is the probability of success, \( \sum_{i=0}^{n} q_i \) is the fail probability and \( \sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1 \).

Given \( P = (p_1, p_2, ..., p_n) \) and \( Q = (q_0, q_1, q_2, ..., q_n) \), the optimal BST is that for which the medium time to solve the search operations has the minim value. In the perspective of medium time computing, the BST is inflated with a set of pseudo-vertices corresponding to the failure intervals. The pseudo-vertices become the new leaves of the BST. A pseudo-vertex \( e_i \) will correspond to the search operations for values that belong to the interval \( (a_i, a_{i+1}) \) (see Fig. 1).

The time cost for a such tree is the medium time for solving a searching operation i.e:

\[
\text{cost}(T) = \sum_{i=1}^{n} p_i \ast \text{level}(a_i) + \sum_{i=0}^{n} q_i \ast (\text{level}(e_i) - 1)
\]