Self-tuning Reactive Distributed Trees for Counting and Balancing

Phuong Hoai Ha, Marina Papatriantafilou, and Philippas Tsigas

Department of Comp. Science, Chalmers University of Technology, SE-412 96 Göteborg, Sweden
{phuong, ptrianta, tsigas}@cs.chalmers.se

Abstract. The main contribution of this paper is that it shows that it is possible to have reactive distributed trees for counting and balancing with no need for the user to fix manually any parameters. We present a data structure that in an on-line manner balances the trade-off between the tree traversal latency and the latency due to contention at the tree nodes. Moreover, the fact that our method can expand or shrink a subtree several levels in any adjustment step, has a positive effect in the efficiency: this feature helps the self-tuning reactive tree minimize the adjustment time, which affects not only the execution time of the process adjusting the size of the tree but also the latency of all other processes traversing the tree at the same time with no extra memory requirements. Our experimental study compared the new trees with the reactive diffracting ones on the SGI Origin2000, a well-known commercial ccNUMA multiprocessor. This study showed that the self-tuning reactive trees i) select the same tree depth as the reactive diffracting trees do; ii) perform better and iii) react faster.

1 Introduction

Distributed data structures suitable for synchronization that perform efficiently across a wide range of contention conditions are hard to design. Typically, “small”, “centralized” such data structures fit better low contention levels, while “bigger”, “distributed” such data structures can help in distributing concurrent processor accesses to memory banks and in alleviating memory contention.

Diffracting trees \cite{1} are distributed data structures. Their most significant advantage is the ability to distribute a set of concurrent process accesses to many small groups locally accessing shared data, in a coordinated manner. Each process(or) accessing the tree can be considered as leading a token that follows a path from the root to the leaves. Each node is a computing element receiving tokens from its single input (coming from its parent node) and sending out tokens to its outputs; it is called balancer and acts as a toggle mechanism which, given a stream of input tokens, alternately forwards them to its outputs, from left to right (sending them to the left and right child nodes, respectively). The result is an even distribution of tokens at the leaf nodes. Diffracting trees have been introduced for counting-problems, and hence the leaf nodes are counters, assigning numbers to each token that exits from them. Moreover, the number of tokens that are output at the leaves, satisfy the step property, which states that: when there are no tokens present inside the tree and if \( \text{out}_i \) denotes the number of tokens that have been output at leaf \( i \), \( 0 \leq \text{out}_i - \text{out}_j \leq 1 \) for any pair \( i \) and \( j \) of leaf-nodes such that \( i < j \) (i.e.
if one makes a drawing of the tokens that have exited from each counter as a stack of boxes, the combined outcome will have the shape of a single step).

The fixed-size diffracting tree is optimal only for a small range of contention levels. To solve this problem, Della-Libera and Shavit proposed the reactive diffracting trees, where each node can shrink (to a counter) or grow (to a subtree with counters as leaves) according to the current load, in order to attain optimal performance [2]. The algorithm in [2] uses a set of parameters to make its decisions, namely folding/unfolding thresholds and the time-intervals for consecutive reaction checks. The parameter values depend on the multiprocessor system in use, the applications using the data structure and, in a multiprogramming environment, on the system utilization by the other programs that run concurrently. The programmer has to fix these parameters manually, using experimentation and information that is commonly not easily available (future load characteristics). A second characteristic of this scheme is that the reactive part is allowed to shrink or expand the tree only one level at a time, making the cost of a multi-adjustment phase on a reactive tree become high.

In this work we show that reactiveness and these two characteristics are not tied together: in particular, we present a tree-type distributed data structure that has the same semantics as the reactive trees that can expand or shrink many levels at a time, without need for manual tuning. To circumvent the need for manually setting parameters, we have analyzed the problem of balancing the trade-off between the two key measures, namely the contention level and the depth of the tree, in a way that enabled the use of efficient on-line methods for its solution. The new data structure is also considerably faster than the reactive diffracting trees, because of the low-overhead, multilevel reaction part: the new reactive trees can shrink and expand many levels at a time without using clock readings. The self-tuning reactive trees, like the reactive diffracting trees, are aimed in general for applications where such distributed data structures are needed. Since the latter were introduced in the context of counting problems, we use similar terms in our description, for reasons of consistency.

The rest of this paper is organized as follows. Section 2 presents the key idea and the algorithm of the self-tuning reactive tree. Section 3 describes the implementation of the tree. Section 4 presents an experimental evaluation of the self-tuning reactive trees, compared with the reactive diffracting trees, on the Origin2000 platform, and elaborate on a number of properties of our algorithm. Section 5 concludes this paper. Due to the space constraint, the correctness proof of our algorithm is presented in [3].

2 Self-tuning Reactive Trees

2.1 Problem Description

The problem we are interested in is to construct a tree that satisfies the following requirements:

---

1 We do not use term diffracting in the title of this paper since our algorithmic implementation does not use the prism construct, which is in the core of the algorithmic design of the (reactive) diffracting trees.