Dynamic Pipelining of Multidimensional Range Queries*

Amalia Duch, Daniel Lugosi, Edelmira Pasarella, and Cristina Zoltan

Universitat Politècnica de Catalunya, Spain
{duch,edelmira,zoltan}@cs.upc.edu

Abstract. The problem of evaluating orthogonal range queries efficiently has been studied widely in the data structures community. It has been common wisdom for several years that for queries containing more than 20% of the elements of the dataset a linear scanning of the data was the most efficient solution. In recent experimental works using modern hardware–with main memory and parallelism– the conclusion is that linear scan is preferable for almost every query configuration (even containing a 1% of the data). In this work we propose an alternative approach to evaluate multidimensional range queries based on the dynamic pipeline paradigm –using main memory and concurrency. Our aim is to prove that under this framework, it is possible to beat the performance of linear scanning by the one of hierarchical multidimensional data structures– such as kd trees, quad trees, R trees or similar.

1 Introduction

It is a common task nowadays to ask Google Maps for the closest gas station or TripAdvisor for good restaurants around a specific area. These requests are examples of a computing task –frequent in a wide range of applications– that is formally called the *Associative Retrieval* problem [2, 4]. In associative retrieval we consider a collection $\mathcal{F}$ of n records. Each *record* (or key) is an ordered k-tuple ($k \geq 2$) $x = (x_1, \ldots, x_k)$ of values (the attributes or coordinates of the record) drawn from domain $D = \prod_{1 \leq j \leq k} D_j$, where each D_j is totally ordered.

A *range query* over $\mathcal{F}$ is the retrieval of those of its records that fall inside a given region. Specifically, we consider *orthogonal* range queries Q, in which the region is specified by a sequence of s unidimensional ranges, this is, $Q = (i_1, l_1, u_1), \ldots, (i_s, l_s, u_s)$, where $1 \leq s \leq k$, $i_j \neq i_{j'} \forall j \neq j'$ (with $1 \leq j \leq s$ and $1 \leq j' \leq s$) and every triplet (i_j, l_j, u_j) fix the lower (l_j) and upper (u_j) boundaries of the unidimensional range for coordinate i_j ($1 \leq i_j \leq k$). If $s = k$ then we say that Q is a *complete* range query. Otherwise, we say that it is *partial*.

In order to efficiently deal with orthogonal range queries the storage of the records in $\mathcal{F}$ should be crucial. Extensive collections of general purpose multidimensional data structures –such as kd-trees, quad-trees or R-trees– have been proposed theoretically as adequate storage methods [2, 4] to support range

* Work supported by grant GRAMM (TIN2017-86727-C2-1-R) and EU FEDER funds.

queries. However, in practice, the usefulness of this approach heavily relies on the selectivity and configuration of the sequence of range queries and, unfortunately common wisdom told that a simple scan beats multidimensional data structures for queries accessing more than 15%-20% of a data collection [5].

Recently, multidimensional range queries as well as the efficiency of hierarchical multidimensional data structures to support them have been revisited under a modern hardware perspective [5]. Moreover, in [5] the authors state that in current machines –using main memory and parallelisation– data structures are useless even for very selective range queries an thus, they conclude that in current machines scanning should be favoured over parallel versions of such data structures.

In this work, we propose a new way to parallelise the multidimensional range query problem using the *dynamic pipeline* model [1, 3]. Our aim is to prove that, with our algorithm, the use of hierarchical multidimensional data structures would be preferable over scanning for range queries containing a sub-linear number of elements of the collection.

2 Dynamic Pipeline Algorithms

We propose an algorithm based on a dynamic pipeline [1, 3] of processes via an asynchronous model of computation, synchronised by means of channels. In general, a *dynamic pipeline* is a data-driven unidimensional and unidirectional chain of stages connected by means of data channels. This computational structure stretches and shrinks depending on the spawning and the lifetime of its stages. A dynamic pipeline is similar to an ordinary pipeline, except that the number of stages that it contains is not fixed but dynamically generated at runtime. In fact, it is self-adaptive to the characteristics of a specific query.

Algorithms under this paradigm must specify four kind of stages: *input*, *output*, *generator* and *filter* stages as well as the number and the type of the I/O unidirectional channels. The input and output stages are the interface of the pipeline, managing the input and output data respectively. Input data is fed to the input stage and the output stage will produce results. The generator is responsible to create the (parameterised) filter stages.

We now describe two algorithms to solve range queries based on the dynamic pipeline model: a naïve algorithm equivalent to a linear scan of the whole dataset followed by the algorithm that we propose, based on a preprocessing of the dataset by means of data structures such as kd trees, quad trees or R trees. We will describe both algorithms for a single range query since the same process is applicable to a sequence of queries iterating on the process for a single one.

Naïve Algorithm. We start by describing a naïve algorithm equivalent to a concurrent approach of the complete scan of the data set.

To answer Q using the pipeline approach it is necessary to have a recursive process that constructs a sequence of filters (processes). Every filter stands for one of the s unidimensional ranges of Q, let us say j ($1 \leq j \leq s$), and it discards

from further consideration all the points of the data set that are outside range (i_j, l_j, u_j), that is, all those points with i_j-th coordinate smaller than l_j or greater than u_j. Since the query has s ranges, the pipe will end up with s processes acting concurrently.

This naïve algorithm starts by setting an initial pipeline consisting of 3 stages –the input, the generator and the output stages, in this sequential order– and two channels –the first carry the sequence of triplets of Q and the second the sequence of points in $\mathcal{F}$.

The process starts by feeding (in sequential order) the input stage with the triplets of Q carried by the first channel. The configuration of the pipe evolves (stretches) as follows. The input stage passes the data from the first channel (a triplet of the query at a time) to its successor neighbour. At the very beginning the triplet (i_1, l_1, u_1) is passed from the input stage to the generator stage. Every time a triplet arrives to the generator a filter stage, standing for this triplet, is created as the stage immediately previous to the generator stage. So, at the very beginning, a filter f_1 for the first range of Q is created. The pipeline consists now of four stages: input, f_1, generator and output, in this order. When triplet (i_j, l_{i_j}, u_{i_j}) passes through the input stage, it passes also through f_1, since $i_j \neq i_1$, it passes through f_2, ..., f_j, up to arrive to the generator where filter f_{j+1} is created. The process continues until the elements carried by the first channel are all treated and the channel is empty. The pipe now, regarding the initial pipe, has s additional stages, one per each triplet of the query.

The next step is to treat the data carried by the second channel, the points. Every point of the data set passes through the pipe. At every filter f_i, as we already mentioned, if the i-th coordinate of the point is outside the range stored at f_i the point is discarded, otherwise it is passed to next stage. Therefore, all the points that arrive to the output stage should be reported as part of Q.

This naïve algorithm will force to read and check every point in the original set, having therefore complexity proportional to n, independently of the configuration of Q.

Proposed Algorithm. To improve the efficiency of the naïve algorithm we propose a preprocessing of the dataset by means of a hierarchical multidimensional data structure. The data structure can be any of the classical ones [2, 4] (such as kd trees, quad trees, R trees, etc.) with the unique requirement that it divides the domain D of the points into a partition of m k-dimensional hyperrectagles that are called *bounding boxes*, where $m \geq 1$ is the number of elements in the partition and depends on the kind of tree used and the number of levels of that tree. Each bounding box BB is defined by a sequence of k ranges, this is, $BB = (1, l_1, u_1), \ldots, (k, l_k, u_k)$. In our preliminar experiments we use quad tries [2, 4] to preprocess the data points and to end up with a sequence $S = BB_1, \ldots, BB_m$ of bounding boxes. It is worth noting that the dynamic pipeline algorithm works identically with any other multidimensional data structure that fits the previous requirement.

Now, instead of directly filtering points, the pipe will filter, first, the sequence S of bounding boxes produced by the data structure (it will have then 3 channels

instead of two, as before). The bounding boxes will pass through the pipe by their specification and not by the points that they contain. Every filter stage, checking for the i-th range of the query, will discard from further consideration all the bounding boxes whose i-th range does not intersect the i-th range of the query and will pass the intersecting bounding boxes to next stage. Additionally, the filtering of bounding boxes divides them into two groups: BBC (the group of bounding boxes which are completely contained into Q) and BBP (the group of bounding boxes that intersect Q but are not contained in it).

The algorithm will output all points that are inside of BBC bounding boxes (since they are all in Q) and it will look and filter all the points in BBP bounding boxes to decide whether they are in Q.

The total number of treated points corresponds to the number of points contained in the BBC and BBP bounding boxes, which can be considerably less than the total number of points in $\mathcal{F}$ (but this highly depends on the configuration of the queries, data points, and chosen data structure). Additionally, the algorithm incurs in the cost of checking the m bounding boxes of S. Our proposal is to maintain this cost negligible compared to the number of points that have to be checked by choosing correctly the number of levels of the tree data structure during the preprocessing of the data.

3 Ongoing Work

We have implemented quad tries in the C++ programming language producing with this program the sequence of m files that containing the points inside the corresponding bounding box in sequence S. Besides, we have implemented the pipeline in the Go programming language (because of its mechanisms of go-routines and channels). Our preliminar experiments show that our proposed algorithm beats the naïve one treating systematically half the points of the data set for queries containing up to a 25% of the points of $\mathcal{F}$. We plan to conduct further experiments according to the following guidelines: (a) Considering huge datasets and allocating their corresponding (tree-like) hierarchical representations in the RAM, we envision that the performance of our algorithm overcomes the results presented in [5] under similar conditions and thus, it overcomes the complexity of linear scanning. We will study, then, the incidence of stressing the population of the memory in order to find insights regarding the percentage of memory that can be used for storage purposes without affecting the performance of the schedule and the memory management of the Go system; (b) Under the premise that the set of points is uniformly distributed, we plan to measure the incidence of the chosen level of the data structure (and thus of the number m of bounding boxes to be filtered) on the performance of our algorithm; (c) The dimensionality of the data increases the parallelism of our algorithm –which depends on the query– so we are interested in studying how, eventually, our algorithm is more suitable than other proposals in high dimensional settings; (d) Finally, in order to study the scalability and real applicability of our model, we plan to conduct our experiments with big real datasets and benchmarks.

References

1. J. Aráoz and C. Zoltan. Parallel triangles counting using pipelining. *CoRR*, abs/1510.03354, 2015.
2. V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
3. E. Pasarella, M-E. Vidal, and C. Zoltan. Comparing mapreduce and pipeline implementations for counting triangles. *Electronic proceedings in theoretical computer science*, 237:20–33, 2017.
4. H. Samet. *Foundations of Multidimensional and Metric Data Structures.*
5. S. Sprenger, P. Schäfer, and U. Leser. Multidimensional range queries on modern hardware. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management, SSDBM 2018, Bozen-Bolzano, Italy, July 09-11, 2018*, pages 4:1–4:12, 2018.