

Developing Distributed Data Mining Implementations for a Grid Environment*

Wei Du Gagan Agrawal
Department of Computer and Information Sciences
Ohio State University, Columbus OH 43210
{duw,agrawal}@cis.ohio-state.edu

1 Introduction

Distributed Data Mining (DDM) is the process of analyzing geographically dispersed large datasets for extracting novel and interesting patterns or models [5]. This paper reports our preliminary work in a project that takes a new approach towards distributed data mining. We believe that general grid application development tools and techniques [4] should be used for developing mining implementations that analyze distributed datasets. We use DataCutter, which is a general grid middleware for data intensive problems and is based upon filter-stream programming model [1]. In this paper, we present a case study of developing k-nearest neighbor search using this framework. Our experiments show that filter granularity selection and filter placement are important issues in implementing data mining algorithms with DataCutter. We are also developing language and compiler support to offer a high-level interface for DataCutter. A data parallel dialect of Java is proposed which is suitable for expressing a variety of data mining tasks. This paper also describes our initial ideas for translating a program written in data parallel Java to a filter-stream program that could be run on DataCutter.

2 A Case Study

To validate our belief that distributed data mining implementations can be conveniently developed using DataCutter, we have implemented a distributed version of k-nearest neighbor classifier.

The problem is quite simple: given a 3-dimension range $R = \langle (x_1, y_1, z_1), (x_2, y_2, z_2) \rangle$, and a point $w = (a, b, c)$, where x_1, y_1, z_1 are the coordinates of the upper left corner of R and x_2, y_2, z_2 are the coordinates of the lower right corner of it, we want to find the nearest k neighbors of w within range R , where k is a parameter given to the algorithm. The basic idea for finding k-nearest neighbors in a sequential environment is as follows. For each point $X = (x, y, z)$ in the file, if X is in the range R , we compute the distance from X to w by $dist = \sqrt{(x-a)^2 + (y-b)^2 + (z-c)^2}$, and find the points with the k-lowest values of distance.

In the filter-stream approach, the problem is decomposed

*This work was supported by NSF grant ACR-9982087 and NSF CA-REER award ACI-9733520.

into two filters for local reduction: one for getting all points in the range R , referred to as `range_query`, and the other for selecting the k-nearest neighbors, called `select`. Since data files are distributed at different sites, one more filter is needed for performing global reduction, referred to as `combine`. Communication between filters is implemented via streams. Filter `range_query` outputs all the points of the local data file which lies in R to filter `select` via stream R-S, while filter `select` communicates with `combine` through S-C stream by sending the local nearest k neighbors to the global reduction filter `combine`. Finally, `combine` will decide the globally nearest k neighbors. The following figure shows the filters and their communication streams.

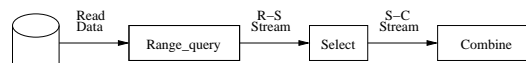


Figure 1. The decomposition of the sequential k-nearest neighbors algorithm into three filters: `range_query`, `select` and `combine`. Arrows show the connectivity among filters via streams.

Several experiments were performed based on different filter placement and filter configuration.

For evaluating the effect of different placement policies, the following two versions of the algorithm are tested: 1. *Place filter `range_query` and `select` on different sites:* As described above, data files are distributed on two sites, say maia and bombur. Each site has about 250,000 3-dimension points, and filter `range_query` is placed on each of them. While filter `select` is placed on another set of sites, say oin and thorin. The filter `combine` for global reduction is placed separately on taygeta. Figure 2 shows this configuration, and 2. *Place filter `range_query` and `select` on the same site:* In this placement scheme, filter `range_query` and `select` are placed on the same site, referred as version 2.

In addition, we also found that the granularity of filters can influence the performance of the algorithm, which is investigated by the following two configurations: 1. *Combine*

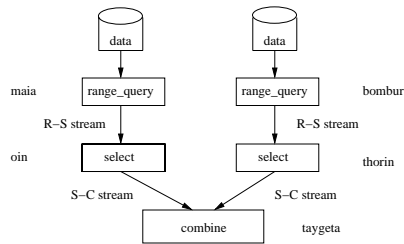


Figure 2. Version 1: Placing filter `range_query` and `select` on different sites

filter range_query and select: Here, the first two filters are combined as one and placed on the site where data files are stored, referred as version 3, and 2. *Combine filter select and combine:* In this version, filter `select` and `combine` are programmed as one filter and put on the remote site from the one where data is stored, referred as version 4.

The running time is given in Table 1 for the above four versions of k -nearest neighbors algorithm where $k=8$ and $k=20$. It's easy to understand why version-3 performs the best and version-2 is better than the remaining two versions. All the data falling in the range R need to be transmitted from filter `range_query` to `select` via a stream. While those two filters are put together on the same site, communication cost over the network are sharply reduced, especially for larger k . Furthermore, if their functions are to be accomplished by only one filter, the cost for copying from one buffer to another between filters is also saved, which gives the best performance over the four versions.

Overall, our experience in implementing k -nearest neighbor search using DataCutter has shown that filter granularity and filter placement are critical for achieving high-performance.

	version-1	version-2	version-3	version-4
$K = 8$	83366	70655	58796	87987
$K = 20$	1000548	874385	122045	1137217

Table 1. Running time for 4 versions of k -nearest neighbors algorithm, where $k=8$ and $k=20$

3 Language and Compiler Framework

We propose to use a data parallel dialect of Java that we had previously used for the same class of applications on a cluster environment [3]. In this dialect of Java, a programmer can specify a computation assuming that all data is available in a flat memory and all computations are done on a single processor.

Our on-going research targets at translating the language interface we described to filter-stream based programs that can be executed on DataCutter. Consider any data parallel loop, the only loop-carried dependencies possible are on data members of an object implementing a reduction interface. Further, these dependencies can only arise in self-updates using associative and commutative operations. Therefore, a data parallel loop can be executed independently on sites having portions of datasets, and a global combination function can be applied later to obtain the final results.

We represent the data parallel loop using the Static Single Assignment (SSA) form [2]. Each assignment or conditional statement in the resulting code, enclosed in the loop body, represents a potential filter. Our filter enumeration phase ends by listing all potential filters.

After the filter enumeration phase, we can represent the program as a graph. A cost is associated with each node and edge. A node simply represents the processing cost of the filter. An edge represents the data transfer cost. Our goal is to minimize the overall execution time on a given set of computing resources.

Optimally, this process will involve the following: 1) enumerate all possible mappings of atomic filters into combined filters, 2) for each such mapping, assign the filters to available computing resources, and find the execution time, and 3) find the composing with the least execution time. But there is no obvious polynomial time algorithm for this task, the only feasible approaches will be based on heuristics. We are developing a number of heuristics for filter granularity selection as part of this project, using the existing work on thread partitioning as our starting point.

4 Summary

This paper has summarized our preliminary work on a new project on distributed data mining. Our project has two novel thrusts. First, we are using a general purpose grid tool for distributed mining implementations. This is in contrast with the existing approaches that rely on specialized systems. Second, we are using compiler technology to provide a high-level language support on top of this middleware.

References

- [1] Michael D. Beynon, Tahsin Kurc, Alan Sussman, and Joel Saltz. Optimizing execution of component-based applications using group instances. In *Proceedings of the Conference on Cluster Computing and the Grid (CCGRID)*, pages 56–63. IEEE Computer Society Press, May 2001.
- [2] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. An efficient method of computing static single assignment form. In *Conference Record of the 16th Annual ACM Symposium on Principles of Programming Languages*, pages 25–35, Austin, TX, January 1989.
- [3] Renato Ferreira, Gagan Agrawal, and Joel Saltz. Compiling object-oriented data intensive computations. In *Proceedings of the 2000 International Conference on Supercomputing*, May 2000.
- [4] Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputing Applications*, 2001.
- [5] Hillol Kargupta, Byung-Hoon Park, Daryl Hershberger, and Erik Johnson. Collective Data Mining: A New Perspective Towards Distributed Data Mining. In Hillol Kargupta and Philip Chan, editors, *Advances in Distributed Data Mining*. AAAI Press, 2000.