PBIRCH: A Scalable Parallel Clustering algorithm for Incremental Data

Ashwani Garg  Ashish Mangla  Neelima Gupta
Vasudha Bhatnagar
Deptt. of Computer Science, University of Delhi, Delhi, India.
{ashgarg16,mangla.ashish}@gmail.com, {ngupta, vbhatnagar}@cs.du.ac.in

Abstract

We present a parallel version of BIRCH with the objective of enhancing the scalability without compromising on the quality of clustering. The incoming data is distributed in a cyclic manner (or block cyclic manner if the data is bursty) to balance the load among processors. The algorithm is implemented on a message passing share-nothing model. Experiments show that for very large data sets the algorithm scales nearly linearly with the increasing number of processors. Experiments also show that clusters obtained by PBIRCH are comparable to those obtained using BIRCH.

1 Introduction

The objective of clustering problem is to partition a population of \( N \) elements each described by \( m \) attributes, into clusters such that elements of a cluster are similar and elements of different clusters are dissimilar [2, 3, 4]. Parallelization is an efficient technique to design fast algorithms to cluster huge data sets.

In this paper we present PBIRCH algorithm, a parallel version of BIRCH [5], which is a widely used algorithm to cluster incremental data. Our algorithm uses data-parallelism with low data dependency and good locality. As a result, it involves less communication. PBIRCH is scalable and speeds-up linearly with the number of processors.

2 BIRCH Algorithm

BIRCH clusters incoming multi-dimensional data points to produce the quality clustering with the available memory and time constraints. It uses the concept of Cluster Feature (CF) to condense information about sub-clusters of points. The Cluster Features are organized in a height-balanced tree called CF-tree. The algorithm makes full use of available memory and requires almost two scans of the input data.

3 PBIRCH - Parallel Birch Algorithm

In this section we present a parallel version of BIRCH algorithm. PBIRCH algorithm runs on the Single Program Multiple Data (SPMD) model using message-passing. Processors communicate with each other through an interconnect. Since communication latencies waste a lot of time, message passing is kept minimum.

Initially, the target data is distributed equally among the processors. Suppose we have \( p \) processors and \( N \) data items. Then each processor gets about \( N/p \) data items. Each processor builds its own CF-tree. When new data items arrive they are distributed cyclically to all the processors. If the data is bursty, as might be the case in several real life applications, the data is divided into \( p \) blocks of roughly equal size and distributed among the processors. Each processor inserts newly allocated data items in its own local CF tree.

After constructing the CF tree, as in case of BIRCH, a clustering algorithm is used to cluster the CFs at the leaf nodes whenever required. However, in the present context we need a parallel clustering algorithm to accomplish this task. Since the number of CFs at the leaf nodes of the CF trees is much less than the number of data items, parallel \( k \)-means algorithm is a good choice [1]. Initially, one processor chooses \( k \) initial seeds for the \( k \)-means algorithm, and broadcasts it to all the processors. Each processor uses the seed to cluster its CFs locally. The means of the local clusters are then exchanged using all-reduce and the global means are computed. This is the only step that requires data exchange among processors in every iteration.

At this point each processor has the \( k \) global means. Starting with these \( k \) means each processor recomputes the clusters and the procedure repeats. The algorithm terminates when the change in clusters between consecutive iterations becomes less than some specified threshold.

We believe that since the number of CFs (at leaf nodes) will be more in parallel than that in sequential, more representatives of data items are available and one should expect clusters of better quality.
The main steps of the PBIRCH algorithm are:

1. Data distribution: Initially data is divided equally among all processors. When new data items arrive they are distributed cyclically (block cyclically) to all the processors.

2. Computing the CF trees concurrently: Initial in-memory CF tree is generated by each processor using local data. If a tree runs out of memory at any point of time then it is rebuilt by increasing the threshold [5].

3. Apply clustering on the leaf nodes: Apply a parallel clustering algorithm on the CFs stored at the leaf nodes.

Experiments further revealed that the sum of radii of the clusters obtained by running the parallel algorithm is slightly lesser than that of the clusters obtained by running BIRCH. This supports our belief that the cluster quality might improve in the parallel version (Section 3).

5 Conclusion

In this paper, we presented PBIRCH, a parallel version of BIRCH algorithm for clustering massive data sets. Our algorithm uses data-parallelism and involves negligible communication overheads. The algorithm runs on shared-nothing message passing architecture. It scales well with the increasing size of the data. The algorithm speeds up nearly linearly with the increasing number of processors. In addition to obtaining time gain, we also show that the quality of the clusters also improves in the parallel version.

References