Parallel Simultaneous Co-clustering and Learning with Map-Reduce

Meghana Deodhar, Clinton Jones and Joydeep Ghosh
Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712
Email: {deodhar@ece, jones@mail, ghosh@ece}.utexas.edu

Abstract—Many data mining applications involve predictive modeling of very large, complex datasets. Such applications present a need for innovative algorithms and associated implementations that are not only effective in terms of prediction accuracy, but can also be efficiently run on distributed computational systems to yield results in reasonable time. This paper focuses on predictive modeling of multirelational data such as dyadic data with associated covariates or “side-information”. We first give illustrative examples of applications that involve such data and then describe a general framework based on Simultaneous CO-clustering And Learning (SCOAL), which applies a divide-and-conquer approach to data analysis. We show that the main elements of the SCOAL algorithm can be effectively parallelized using the Map-Reduce framework. Experiments on Amazon’s EC2 demonstrate that the proposed parallelizations result in considerable improvements in run time when using a cluster of machines.

Keywords—Predictive modeling, distributed data mining, dyadic data, Map-Reduce

I. INTRODUCTION

Several industry scale applications involve very large datasets. For instance, Google, Amazon and Yahoo! process several terabytes of data daily. Due to the relatively low cost of hardware, it is increasingly attractive to parallelize computations across hundreds of CPUs to scale up to the data volume. Consequently, a lot of attention has recently been devoted to approaches for distributed processing of large-scale data.

In this paper we focus on predictive modeling of data with a specific structure, namely dyadic (bi-modal) data [3], which is now pervasive in a variety of real life applications. Dyadic data consists of measurements on dyads, which are pairs of elements from two different sets (modes). The measurements can be represented as the entries of a matrix, whose rows and columns are the two sets of elements. The independent variables, also referred to as covariates, are associated with the entities along the two modes. For example, consider the dataset of ratings of a set of users for a set of movies. The independent variables are (i) attributes associated with each user, e.g., age, gender, income, (ii) characteristics of each movie, e.g., genre, release year, number of awards, and (iii) attributes associated with a (user, movie) pair, e.g., whether the user’s favorite actor stars in the movie. The data can be thought of as a “users by movies” matrix with the response variable, i.e., the ratings, represented by the cells of the matrix.

The following are examples of industry applications that involve predictive modeling of large-scale dyadic data:

Online ad CTR prediction. A key goal of search engines such as Yahoo! and Google is to maximize the click-through-rate (CTR) on online ads by serving users the ads they are most likely to click on. The massive scale of the ad targeting problem and its obvious business relevance has recently attracted attention from the data mining community (see [5], [2], [1]). The data arising in this application is inherently dyadic, with users and ad categories representing the two sets of entities. An ad category is a specific topic of interest, e.g., loans, travel, parenting and children. These categories are annotated by attributes such as descriptive keywords, historic CTR and volume, while users are also described by features such as demographics, geographical location and metrics based on previous browsing behavior. For some (ad category, user) pairs, the target CTR value is known or easily estimated, and these form the training data. Given a user, the objective is then to select the categories to be served based on the highest predicted CTRs. Such data is very large (typically several hundred million users per week and several thousand categories), very sparse and noisy, with little activity in some low traffic categories. Moreover, the data is very heterogeneous, with widely varying patterns of user behavior across different user and category groups.

Estimating search query-web page relevance. This problem deals with optimizing search engine results by predicting relevance scores of web pages for search queries. Considering the scale of the web and the number of possible query keywords, the amount of data to be modeled is massive.

It is natural for large amounts of data, such as that being modeled by the above applications, to exhibit substantial heterogeneity of predictive patterns across the input space. For example, data on customer-product ratings (such as that maintained by Amazon) exhibits significant heterogeneity across customers that come from a diverse population as well as across products, which span a wide and ever-growing range. It has been shown that in such heterogeneous settings, modeling the response variable with multiple local models, each capturing patterns for a certain region of
the customer/product input space, can yield substantially higher predictive accuracy as compared to a single global model induced from the entire input space [10]. In particular, the recently proposed Simultaneous CO-clustering And Learning (SCOAL) approach [10] provides a versatile and effective framework for predictive modeling of large-scale, heterogeneous, dyadic data. SCOAL addresses the data heterogeneity by representing the data with a collection of multiple localized models, learned concurrently while partitioning the data into local regions. This approach addresses both classification and regression problems, and works with a variety of learning models.

Since SCOAL is a good fit for industry applications like online advertising and product recommendation systems, there is a compelling need to develop a scalable implementation of SCOAL. As described in Section III, several of the operations performed by the SCOAL algorithm are independent and can be carried out concurrently. Consequently, there is massive scope for parallelization in the SCOAL algorithm that can be exploited by a distributed computing environment. An important motivation for developing a parallel version of SCOAL is that since it distributes computation across multiple machines it is not necessary to have all the data on a single machine, and therefore the algorithm can scale to much larger datasets than a serial SCOAL implementation.

In this paper we develop a parallelized SCOAL implementation (Parallel SCOAL) based on the Map-Reduce [9] paradigm, using Hadoop (hadoop.apache.org). Hadoop is an open source java (Apache) implementation of Map-Reduce. We empirically evaluate the accuracy and scalability of Parallel SCOAL and provide guidelines for performance tuning on real-world movie recommendation datasets.

II. BACKGROUND

A. Simultaneous Co-clustering and Learning (SCOAL)

To build the necessary background, in this section we briefly describe the SCOAL framework.

Data Representation. Consider dyadic data involving two sets of entities, generically denoted as “customers” and “products”. The response variable values can be represented as the entries of an \( m \times n \) matrix \( Z \) of customers and products, with cells \( z_{ij} \) representing the corresponding customer-product preferences, e.g., the rating customer \( i \) gives for product \( j \). We denote the independent variables (covariates) associated with rating \( z_{ij} \) by vector \( x_{ij} \). The covariate vector can be partitioned into customer attributes, \( e_i \) and product attributes, \( p_j \), as well as annotations associated with a (customer, product) pair, \( a_{ij} \), i.e., \( x_{ij}^T = [e_i^T, p_j^T, a_{ij}] \). 

SCOAL [10] is an efficient and versatile framework that provides a way to model data of the nature described above by a collection of local models. The key idea is to partition the data matrix into a grid of blocks (co-clusters) such that each co-cluster can be well characterized by a single predictive model that relates the independent variables to the response variables in the co-cluster.

Formally, let \( k, l \) represent the number of row and column clusters, and let \( \rho, \gamma \) be mappings from the \( m \) rows to the \( k \) row clusters and from the \( n \) columns to the \( l \) column clusters respectively. The missing values in matrix \( Z \) are handled by associating a weight \( w_{ij} \) with each cell \( z_{ij} \). The weights of the known (training) matrix cell values can be set to 1, while missing values can be ignored during the modeling process by setting their weights to 0. In general, the weights can take other non-negative values to reflect uncertainties associated with the responses. Considering a regression problem, let us start with linear generative models for the data, for ease of exposition. The aim of SCOAL is to find a co-clustering defined by \((\rho, \gamma)\) and the associated \( k \times l \) regression models that minimize the following objective function:

\[
\sum_{g=1}^{k} \sum_{h=1}^{l} \sum_{u, \rho(u) = g, \gamma(v) = h} w_{uv} (z_{uv} - \hat{z}_{uv})^2, \tag{1}
\]

\[
\hat{z}_{uv} = \beta_{gh}^T x_{uv}. \tag{2}
\]

A suitable co-clustering \((\rho, \gamma)\) can be obtained by an algorithm that iterates over three steps: (i) row cluster update, (ii) column cluster update, and (iii) model update. The row cluster update step greedily assigns each row to the row cluster that minimizes the squared error for that row. The column cluster update step is analogous to the row cluster update. In the model update step, the \( \beta \) for each co-cluster is the solution to a least squares problem. Each step is guaranteed to reduce the objective function, so the algorithm converges to a locally optimal solution. For more details on the SCOAL algorithm and the pseudo-code, the reader is referred to [10]. After the algorithm converges, the co-cluster assignments and their respective regression models can be used to predict unknown customer-product preference values. SCOAL is readily extensible to other predictive models as well as to classification problems by modifying equation (1) and/or equation (2) [10]. In addition, SCOAL can also be applied to multi-modal data with covariates.

B. The Map-Reduce Paradigm

Map-Reduce, an effective paradigm for coordinating parallel computing, has attracted attention from both academia and industry. It provides a clean and easy to use distributed programming abstraction. Map-Reduce is described in the original paper by Dean and Ghemawat [9]. The Map-Reduce paradigm primarily draws from functional programming. Any computation under this paradigm is decomposed into a Map function and a Reduce function. The inputs and

\[1\]Note that up to two of these three vectors \((e_i, p_j, a_{ij})\) could be empty.

\[2\]Generalization to other models is discussed in [10].
outputs of the two functions are (key, value) pairs, denoted as \(<k,v>\). The functions are specified as follows:

Map: \(<k_{\text{in}}, v_{\text{in}}> \rightarrow <k_{\text{int}}, v_{\text{int}}>\)
Reduce: \(<k_{\text{int}}, v_{\text{int}}> \rightarrow <k_{\text{out}}, v_{\text{out}}>\)

Records from the data source, e.g., from files or from a database, are sent to the Map function as (key, value) pairs. The Map function produces an intermediate key and associated value \(<k_{\text{int}}, v_{\text{int}}>\) from the input. After the Map operation is done, all the intermediate values for a given intermediate key are merged into a list. The Reduce function takes as input the intermediate key and its corresponding list of values denoted as \(<v_{\text{int}}>\). The intermediate values are then combined, finally resulting in an output (key, value) pair \(<k_{\text{out}}, v_{\text{out}}>\). Multiple Map/Reduce jobs can be executed in parallel. However, there is a bottleneck at the shuffle operation that groups together the intermediate values associated with the same key after the Map phase to prepare the input to the Reduce phase. Hence, the Reduce phase cannot start until the Map phase is completely finished.

III. PARALLEL SCOAL ALGORITHM

In this section we present our design for a parallel implementation of SCOAL using Map-Reduce. Recall that the SCOAL algorithm iterates over three steps until convergence: (i) row cluster re-assignment, (ii) column cluster re-assignment and (iii) model building (i.e., learning prediction models in each co-cluster.) The independence of the several operations performed in each step makes massive parallelization possible. A major computational bottleneck in the serial execution of SCOAL is the computation of row and column distances from their respective cluster candidates, which is then used by the cluster assignment steps. However, one can observe that the distance can be computed independently for each cell of the data matrix. The row or column distance can then be computed by appropriately grouping together entries in the same row or column. This forms the basis of the parallel SCOAL algorithm.

Each entry in the data matrix, along with its covariate vector, weight, row/column cluster assignment and other meta-data is bundled into a matrix entry tuple. The data can be represented as a collection of entry tuples that are iteratively modified by the three steps of the SCOAL algorithm. A “group by” operation can be done to collect entry tuples for a row/column/co-cluster as required by the three steps. The motivation behind this design is that decomposing the problem by entry tuples (individual data points) is more fine grained than decomposing it by rows or columns as in [13] (described in more detail in Section V), resulting in increased parallelism. The model coefficient vectors \(\{\beta\}\) are maintained as global parameters. They are globally broadcast to the Map-Reduce jobs that use them, i.e., the row and column cluster re-assignments. Figure 1 gives an overview of the parallel SCOAL algorithm. Each iteration is composed of three Map-Reduce jobs chained together, one for each of the three steps described above.

![Figure 1. Overview of the Parallel SCOAL Map-Reduce implementation.](image)

The details of each Map-Reduce job are as follows:

A. Row Cluster Re-assignment

**Map function.** The input to the Map function is a (tuple id, entry tuple) pair. The Map function computes the distance of the entry \(z_{ij}\) from each of the \(k\) possible row clusters that row \(i\) could be assigned to. The row id and an entry-distance tuple (consisting of the matrix entry tuple augmented with the vector of \(k\) distances) are output as the intermediate key and value, respectively.

**Reduce function.** The input to the Reduce function is a row id and the list of values (entry-distance tuples) for the entries in the corresponding row. The Reduce function simply sums up the distances of the individual row elements to find the total distance of the row from \(k\) possible row clusters. The row is then assigned to the row cluster with the minimum distance. The entry tuples are updated with the new row cluster assignment and pairs of tuple ids and entry tuples (without distance vectors) are output.

Figure 2 provides the pseudo-code for the Map and Reduce jobs, assuming linear regression co-cluster models.

B. Column Cluster Re-assignment

The Map and Reduce functions are analogous to the Row Cluster Re-assignment step. The Map function outputs a column id and the entry-distance tuple as the intermediate key and value. The output of the Reduce function is the set of tuple ids and entry tuples associated with a column, updated with the new column cluster assignment.

C. Co-cluster Model Building

**Map function.** The input to the Map function is a (tuple id, entry tuple) pair. The function itself is an identity function, which simply emits the co-cluster id as the intermediate key and the entry tuple as the intermediate value.
Algorithm SCOAL \(k=5, l=5\)

SCOAL \(k=4, l=4\)

0.856 (0.001)

Global \(k=1, l=1\)

Global \(k=1, l=1\)

1.194 (0.001)

1.192 (0.005)

\[\begin{align*}
\text{Map}(\text{TupledId t_id, Tuple t}) & \\
& \text{for each row cluster } i = 1 : k \\
& \quad \text{dist}_\text{vec}(i) = (t.z - \beta_i^T.t.x)^2 \\
& \quad \text{Emit}(t\_\text{row_id}, (t, \text{dist}_\text{vec}))
\end{align*}\]

\[\begin{align*}
\text{Reduce}(\text{RowId row_id, Iterator tuples_list}) & \\
& \text{//initialize distance vector of size } k \text{ to all zeros} \\
& \quad \text{Initialize dist}_\text{vec}(1 : k) = 0 \\
& \text{//aggregate distance vectors over all tuples in the row} \\
& \quad \text{for each } v \in \text{tuples_list} \\
& \quad \quad \text{dist}_\text{vec} + = v.\text{dist}_\text{vec} \\
& \text{//assign row to closest cluster} \\
& \quad \text{[min}_\text{dist}, \text{min}_\text{index}] = \text{min}(\text{dist}_\text{vec}) \\
& \quad \text{new}_\text{cluster} = \text{min}_\text{index} \\
& \text{//update tuple row cluster assignments to new_cluster} \\
& \quad \text{for each } v \in \text{tuples_list} \\
& \quad \quad v.\text{row_cluster} = \text{new}_\text{cluster} \\
& \quad \text{Emit}(v.\text{tuple_id}, v.\text{tuple})
\end{align*}\]

Figure 2. Pseudo-code for the Map-Reduce row cluster re-assignment step.

\[\begin{align*}
\text{Map}(\text{TupledId t_id, Tuple t}) & \\
& \quad \text{Emit}(t.\text{cocluster_id}, t)
\end{align*}\]

\[\begin{align*}
\text{Reduce}(\text{CoclusterId cocluster_id, Iterator tuples_list}) & \\
& \quad \text{//learn co-cluster model} \\
& \quad \beta = \text{LinearRegression(tuples_list)} \\
& \quad \text{obj}_\text{fn} = 0 \\
& \quad \text{for each } t \in \text{tuples_list} \\
& \quad \quad \text{obj}_\text{fn} + = (t.z - \beta^T.t.x)^2 \\
& \quad \text{Emit}(\text{cocluster_id, } (\beta, \text{obj}_\text{fn}))
\end{align*}\]

Figure 3. Pseudo-code for the Map-Reduce co-cluster model building step.

Reduce function. The input to the Reduce function is a co-cluster id and a list of all the entry tuples in the co-cluster. The main job of the Reduce function is to learn the co-cluster model from the entry tuples. In the case of linear regression models, this involves solving a least squares problem and obtaining a new co-cluster-specific \(\beta\) vector. The new model coefficients and the objective function value for the co-cluster are globally broadcast.

A global synchronization occurs after the execution of the three steps, which (i) updates the global \(\{\beta\}\) vectors and (ii) computes the overall objective function value by aggregating over the co-cluster objective function values. If convergence has not been reached, the next iteration is initiated.

IV. EXPERIMENTAL RESULTS

A. Setup

The Parallel SCOAL implementation was evaluated on the distributed computing infrastructure provided by the Amazon Elastic Compute Cloud (EC2: http://aws.amazon.com/ec2/). EC2 provides an easy to use interface to run parallel algorithms on variable-sized clusters of machines along with Hadoop-EC2 configuration and monitoring tools.

Several pre-made and packaged Hadoop EC2 images are also publicly available. Our experiments were carried out on the default small instance EC2 machines with the following specifications: 1.7 GB memory, 1 virtual core, 160 GB instance storage, 32-bit platform and moderate I/O performance.

B. Datasets

We evaluated the Parallel SCOAL framework on the task of predicting user-movie ratings in a recommender system setting. The datasets used for experimentation were two versions of the MovieLens dataset made available by the Grouplens Research Project at the University of Minnesota.\(^3\) MovieLens100K consists of 100,000 ratings (1-5) from 943 users for 1682 movies, while MovieLens1M includes 1 million ratings for 3900 movies by 6040 users. We used a set of 23 attributes including user demographics such as age, gender, employment status and movie date and genre. Both datasets are extremely sparse, with a very small proportion of known user-movie ratings. For MovieLens100K only 6.3% of the matrix entries are known, while for MovieLens1M, 4.26% are known. Both of these datasets are actually small by Hadoop standards, resulting in a large file system overhead compared to actual computation. However, they are sufficient for a preliminary test of the scalability of Parallel SCOAL.

C. Prediction Accuracy

This section evaluates the ability of the SCOAL algorithm to accurately model the heterogeneous, dyadic MovieLens datasets. Table I compares the performance of a single global linear regression model with SCOAL, using linear regression models in each co-cluster and hence representing the data by a collection of linear models. The table displays the test MSE over 10 cross-validation folds for both modeling approaches on the two MovieLens datasets. On both datasets, SCOAL gives significant improvement over a single global model, indicating that multiple local models can better represent the inherent heterogeneity in these datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Algorithm</th>
<th>Test MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovieLens 100K</td>
<td>Global (k=1,l=1)</td>
<td>1.192 (0.005)</td>
</tr>
<tr>
<td></td>
<td>SCOAL (k=4,l=4)</td>
<td>0.915 (0.002)</td>
</tr>
<tr>
<td>MovieLens 1M</td>
<td>Global (k=1,l=1)</td>
<td>1.194 (0.001)</td>
</tr>
<tr>
<td></td>
<td>SCOAL (k=5,l=5)</td>
<td>0.856 (0.001)</td>
</tr>
</tbody>
</table>

Table I: Comparison of global model with SCOAL on MovieLens datasets.

D. Scalability Tests

Figure 4 shows the runtime of Parallel SCOAL on clusters with an increasing number of slave machines\(^4\) on the two MovieLens datasets. The number of Map and Reduce jobs is

\(^3\)The data can be downloaded from http://www.grouplens.org/

\(^4\)A Hadoop cluster is comprised of a single master and multiple slave machines.
set to 10. On both datasets, one can observe that increasing the number of slave machines gives good speed-up, e.g., with two slaves rather than a single one, the running time is cut by half. The runtimes flatten out at around 5 machines, possibly due to an increasing overhead cost of the Hadoop framework, which results in diminishing marginal returns from a larger number of slaves.

![Figure 4. Running time in minutes vs. number of slaves in the cluster.](image)

E. Performance Tuning

While the results presented in Figure 4 serve as a proof of concept of the scalability of Parallel SCOAL, there is scope for further speed-up by tuning several parameters of the Hadoop framework. This section studies the impact of the number of Map and Reduce jobs on performance along with its interaction with the number of machines.

Figure 5 displays the results of varying the number of Map and Reduce jobs on the MovieLens100K dataset on EC2 clusters with 5, 10 and 20 slaves respectively. Under the Hadoop framework, the user can only specify the minimum number of Maps jobs spawned (thus providing a hint to the framework regarding the number of Map jobs), rather than the actual total number. For the results in Figure 5, the number of Map and Reduce jobs executed are equal and vary from 1 to 25, depending on the number of slaves in the cluster.

In contrast, the number of reduce jobs to be spawned can be specified exactly. Figure 6 displays the results of varying the number of Map jobs, while keeping the number of Reduce jobs fixed at 1, on the MovieLens100K dataset on EC2 clusters with 5 and 10 slaves. The actual number of Map jobs spawned are displayed in the figure.

From Figure 5, one can observe that the run time reaches a minimum when the number of Map and Reduce jobs spawned is very close to the number of slave machines. A similar trend is observed in Figure 6. Therefore, a rough guideline is to set the number of Map and Reduce jobs to the number of machines in the cluster, or slightly higher, so as to fully utilize the computing power of all the machines. Moreover, from the run times in the two sets of figures (Figure 5 and 6), one can observe that the parallelization of the Map operations is more critical than that of the Reduce operations. For a fixed number of Map jobs, having more then one Reduce job improves the run time by a relatively small amount. This is due to the fact that the Map jobs do the most expensive operations of computing row/column distances from their respective cluster candidates, which comprises the major computational bottleneck. Because of these expensive operations, tuning the number of Map jobs is more critical to the overall run time.

In addition to the number of Map and Reduce jobs, there are several other options that can be tuned. For instance, another option that impacts performance is the file replication factor. Currently we use the default replication factor of 3. Since our files are small, a larger replication factor could be beneficial, since a larger number of Map jobs can then work on local copies of the data.

V. RELATED WORK

Given the large scale of recent data mining applications, there has been a lot of interest within the data mining community to build scalable and efficient modeling frameworks. The increasing amounts of data being processed have made parallel computation a key component of scalable data mining techniques. For instance, a dataflow implementation of the versatile partitional Bregman co-clustering technique [4], which has several applications in text mining, recommender systems and bioinformatics was shown to give speed-ups of several orders of magnitude over serial implementations on the Netflix problem [7].

The Map-Reduce paradigm is the basis of several distributed algorithms across academia and industry. Several commonly used machine learning algorithms have recently been reimplemented under the Map-Reduce paradigm [6]. This includes a Map-Reduce version of the EM algorithm for Probabilistic Latent Semantic Indexing [8], which has been effectively applied to Google News personalization. Another example is that of DisCo: a distributed co-clustering technique based on Map-Reduce [13]. Grossman and Gu [11] have implemented a high performance data storage and compute cloud to mine large distributed datasets. They demonstrate its effectiveness on a distributed data mining application and conduct comparative studies of their system with the Hadoop Map-Reduce framework. A recent paper by Chen et al. [5] addresses the problem of targeting the most relevant ads to online users, based on historic user behavior. They develop a highly scalable Poisson regression modeling technique using the Hadoop Map-Reduce framework, which enables them to process the entire Yahoo! user base in one day. In contrast to SCOAL, they do not exploit the dyadic data structure, and instead build a separate linear model for each ad category.

The increasing popularity and impact of business applications such as recommender systems, search advertising and market basket analysis has resulted in increased attention from the data mining community on the predictive modeling of dyadic data with covariates [14], [3], [1]. Matrix factorization techniques have also been extended to use additional side-information, e.g., covariates and social network structure, resulting in approximate inference algorithms based on
Variational techniques or Monte Carlo methods [1], [12]. SCOAL differs from these techniques since it takes a local approach that allows for natural parallelization, while all these approaches are global.

VI. CONCLUSIONS AND FUTURE WORK

This paper shows that a general approach to predictive modeling of large multirelational datasets has properties that makes it quite amenable to efficient cloud computing. This demonstrates the suitability of cloud computing for a very large class of data mining applications.

Although the relative speed-up achieved by Parallel SCOAL is good, the absolute runtimes are still high due to the fact that Hadoop itself is slow and the fact that the overhead introduced by HDFS is large. It would be worthwhile exploring other, faster implementations of Map-Reduce such as Phoenix, a shared memory implementation (http://mapreduce.stanford.edu/).

ACKNOWLEDGMENT

This research was supported by NSF grant IIS 0713142.

REFERENCES