ASSIGNMENT 5: Collaborative Filtering
Out: 4/8/02; Due:5/7/02
Programming Parallel and Distributed Systems
Computer Science 178, Spring 2002
Steven P. Reiss

OBJECTIVE

This assignment is designed to first familiarize yourself with MPI and then to gain some experience designing and implementing a parallel application. The actual assignment has two parts. The first part involves implemeting boosting using MPI on a network of workstations. The second part, involves designing and implementing a parallel version of singular value decomposition. Here you will first run it on the network of workstations and then on the IBM SP on campus.

MOTIVATION

Machine learning studies automatic techniques for learning to make accurate predictions based on past observations. A dataset for learning typically consists of a number of items, each of which has several attributes and is labelled with a class label, indicating what kind of item it is. The goal is to build a predictor that, given a new object and its attributes, will predict a class label for it.

A state-of-the-art technique for building predictors is boosting. Boosting is based on the idea of combining many weak (moderately acurate) prediction rules in order to construct a highly accurate one. The boosting algorithm selects a weak prediction rule repeatedly, each time using a different distribution over the data set, and after many rounds the boosting algorithm combines these weak rules into a single prediction rule that hopefully is more accurate than any of the single weak rules. The distribution over the dataset is chosen in a way that the most weight is placed on the items that are most often misclassified by the preceding weak rules; this has the effect of forcing the base learner to focus its attention on the hardest examples. Eventually the selected weak rules are combined into a single rule by a weighted majority vote of their predictions.

In this assignment we will use boosting in an atypical setting, that of information filtering in multi-user systems. In this setting each user may define personalized categories for items, such as movies. In particular, users may annotate items by whether an item is relevant (label 1) or irrelevant (label 0). These preferences or categories will be specific to a particular person, yet there might be similarities between user interests that induce dependencies among the category labels. For example, a movie m_i labeled with $y_i^j \in \{0, 1\}$ by some user u_j might provide evidence about how another user might label this item, especially if the two users have shown similar responses on previous items.

However, typical datasets are extremely large, and in most cases predictions must be generated or updated fast, so it is natural to consider parallelism in the prediction generation process. Moreover, available datasets are very sparse, and algorithms for an approximate reconstruction of the complete dataset are of great importance.

SPECIFICATIONS

PART I

We want to estimate the movie preferences of a single user each time, that we will call *active* user. This means that we want to predict the (hidden) labels of movies according to the active user, where the weak prediction rules in the boosting rounds will be the labels of these movies according to the rest of the users. The data is given in the form of a sparse matrix M, where each row represents a movie, and each column a user profile. An entry $M(i, j) = 1$ denotes that movie i is relevant to user j, and a zero entry that is irrelevant. The standard algorithm for boosting is given in Table 1.

A parallel boosting scheme can be as follows: first the matrix M is split into $m \times (n/k)$ chunks, where k is the number of processors. In a boosting round t, each processor computes steps [4] and [5] of the algorithm on the portion of the matrix they have. Then the processor with the best matching user profile (the processor with the hypothesis $h_{t,k}$ with the smallest ϵ_t), executes steps [6]-[10] and broadcasts the weights w_{t+1} to the rest of the processors for the following boosting round.

PART II

We are given a sparse data set and want to compute a relatively good approximation of the missing data. A simple algorithm for the reconstruction of an incomplete data set can be found in section 5.2 of *"Spectral Analysis of Data"* by Y. Azar et al. This algorithm is based on the singular value decomposition (SVD), so the main task in this part of the assinment is to implement a parallel sparse SVD algorihtm.

Basic background and algorithms for the singular value decomposition of a matrix can be found in *"Matrix Computations"* by G. H. Colub and C. F.

Van Loan. Several methods (subspace iteration method, trace minimization method, Lanczos method and block Lanczos method) for computing the SVD of large sparse matrices on a multiprocessor architecture are presented in *"Large Scale Sparse Singular Value Decomposition"* by M. W. Berry. You can pick a method of your choice, but one of the methods described in the above paper is recommended due the nature of the data matrices we are dealing with in the collaborative filtering setting.

There are copies of above papers in the directory `/course/cs178/handouts/`.

TESTING

PART I

You are given a fully reconstructed dataset that contains 1682 movies and 943 user profiles (`/course/cs178/handouts/MovieLensBinary.dat`). The matrix is stored columnwise, that is the movie votes for each user are stored in each line.

A user should be randomly selected to be the active user, and his/her profile put aside. The profiles of the rest of the users will comprise the set of weak prediction rules for boosting. Moreover, for all users the movie votes should be split into a training set (90%) and a test set (10%). The parallel boosting algorithm should be trained on the training set and tested for the active user on the test set. Average accuracy over 100 repetition of the experiment, where a user is randomly selected to be the active user, and running times for the experiments should be reported.

PART II

You are given a sparse dataset that contains 1623 movies and about 61265 user profiles. (`/course/cs178/handouts/EachMovieBinary.dat`) The sparse matrix is stored columnwise, that is the movie ids followed by a delimiter and the vote (`<mid>:<vote>`) of the nonzero entries for each user are stored in each line. Note that here 1 denotes that a movie is relevant to the user, -1 that it is irrelevant, and 0 denotes a missing vote.

The full matrix should be reconstructed for increasing number of user profiles (see how much you can deal with), and the reconstruction times should be reported. Moreover, part I should be repeated for the various reconstructed matrices and accuracy results reported.

MECHANICS

Your MPI program for the first part of the assignment should run on a network of workstations. A recent implementation of MPI is available in `/cs/src/mpi/lam` on the Suns. The commands in `/cs/src/mpi/lam/bin` starting with "`lam`" can

Input: zero-one $m \times n$ matrix M (votes matrix)
zero-one $m \times 1$ vector V (active user's votes)
a constant T

[1] **initialize for all** $i : w_1(i) := 1/m$

[2] **for** $t = 1$ **to** T **do**

[3] **for all** $i : p_t(i) := w_t(i)/(\sum_i w_t(i))$

[4] $h_t := \arg\min_j \sum_i p_t(i)[\![M(i,j) \neq V(i)]\!]$

[5] $\epsilon_t := \sum_i p_t(i)[\![M(i,j) \neq V(i)]\!]$

[6] **if** $\epsilon_t > 1/2$ **then**

[7] $T := t - 1$

[8] **goto 13**

[9] **end if**

[10] $\beta_t := \epsilon_t/(1 - \epsilon_t)$

[11] **for all** $i : w_{t+1}(i) := w_t(i)\beta_t^{1-[\![M(i,h_t)\neq V(i)]\!]}$

[12] **end for**

[13] **Output:** $H(i) = \arg\max_{y\in\{0,1\}} \sum_{t=1}^{T} (\log 1/\beta_t)[\![M(i,h_t) = y]\!]$

Table 1: The sequential boosting algorithm. The formula $[\![E]\!]$ is 1 if E is true and 0 otherwise.

be used to set up the set of machines to be included in the network of workstations. The `lamboot` command takes one argument, a file containing the host names of the machines to run on. The command `mpirun -np <#> [command]` can then be used to run your program (command) using # processors. Include files are in `/cs/src/mpi/lam/include` and the libraries you will need to bind with are in `/cs/src/mpi/lam/lib`. (You might also need `-lsocket` on the suns). You probably want to get your program working with one processor initially. And then move to multiple processors.

Your second part of the assignment should also run on the SP on campus, using your TCASCV account. You might find *"Getting Started on the IBM SP"* (`http://www.cascv.brown.edu/spuser.html`) useful as an introduction to computing resources at TCASCV.