2010 IEEE International Conference on Data Mining Workshops

Parallelized Boosting with Map-Reduce

Indranil Palit*, Chandan K Reddy†
Department of Computer Science
Wayne State University, Detroit, Michigan 48202
*Email: *indranilpalit@wayne.edu, †reddy@cs.wayne.edu*

***Abstract*—In this paper, we propose two novel algorithms, ADABOOST.PL (Parallel ADABOOST) and LOGITBOOST.PL (Parallel LOGITBOOST), that facilitate simultaneous participation of multiple computing nodes to construct a boosted classifier. Our algorithms can induce boosted models whose generalization performance is close to the respective baseline classifier. By exploiting their own parallel architecture both the algorithms gain significant speedup. We used the *Map-Reduce* framework to implement our algorithms and experimented on a variety of synthetic and real-world data sets to demonstrate the performance in terms of classification accuracy, speedup and scaleup.**

I. INTRODUCTION

In several scientific and business applications, it has almost become a common practice to gather information which typically contains millions of training examples with thousands of features. Furthermore, data is either generated or gathered everyday at an unprecedented rate. To efficiently handle such large-scale data, faster processing and optimization is becoming more important. Hence, it has become vital to develop new algorithms that are more suitable for parallel architectures. One simple approach could be to deploy a single inherently parallelizable data mining program to multiple data on multiple computers. But, for algorithms that are not inherently parallelizable in nature, redesigning to achieve parallelization is the alternative.

Ensemble classifiers [1], [2], [3] are a powerful set of learners in data mining that use multiple models to obtain better predictive performance compared to other methods [4]. Boosting is a powerful ensemble method with proven theoretical performance guarantees. In the case of boosting, it becomes tricky to parallelize because of the sequential nature of the algorithm. Though not algorithmically constrained, most boosting algorithms iteratively learn weak classifiers with respect to a distribution and add them to a final strong classifier. Thus, weak learners in next iterations focus more on the examples that previous weak learners misclassified. However, this dependent iterative setting of boosting makes it inherently a serial algorithm. The task of making iterations independent of each other and thus leveraging boosting for parallel architectures is non-trivial and demands some research attention.

In this paper, we address the problem of parallelizing boosting from a different perspective. We propose two novel algorithms, ADABOOST.PL (Parallel ADABOOST) and LOGITBOOST.PL (Parallel LOGITBOOST), that achieve parallelization in both time and space. Parallelization in space is also important because of the limiting factor posed by the memory size. Large data sets, that can not fit into the main memory, are often needed to swap between the main memory and the (slower) secondary storage, introducing latency cost which sometimes may even diminish the speedup gained by parallelization in time. Both the proposed algorithms are designed to work in cloud environments where each node in the computing cloud works only on a subset of the whole data. *The combined effect of all the parallel working nodes is a boosted classifier model induced much faster and with an excellent generalization capability.*

We empirically show that, while maintaining a competitive test accuracy, the algorithms achieves significant speedup compared to respective baseline (ADABOOST or LOGITBOOST) implemented in a single machine. For comparison purposes, we also experimented with MULTBOOST [5], which is a variation of ADABOOST capable of being fitted in a parallel architecture; and showed that ADABOOST.PL performs better than MULTBOOST both in terms of prediction accuracy, speedup and scaleup. For the implementation, we used *Map-Reduce* [6] framework, which is a simple model for distributed cloud computing.

II. ADABOOST.PL

In this section, we describe our proposed algorithm ADABOOST.PL. Before that, we would like to discuss ADABOOST [7] in brief. The pseudocode for ADABOOST is described in Algorithm 1. Let the data set $D_n = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, where each example $x_i = (x^1, x^2, ..., x^d)$ is a vector with d attribute values and each label $y_i \in \{+1, -1\}$. The algorithm assigns weights $w^t = \{w_1^t, w_2^t, ..., w_n^t\}$ for all the examples in D_n, where $t \in [1, T]$ and T is the total number of boosting iterations. Before starting the first iteration these weights are uniformly initialized (line 1) and they are updated in every consecutive iteration (lines 7-10). It is important to note that, for all t, $\sum_{i=1}^{n} w_i^t = 1$. At each iteration, a weak learner function is applied to the weighted version of the data which then returns an optimal weak hypothesis $h^{(t)}$ (line 3). At each iteration, a weight (α^t) is assigned to the weak classifier (line 5). At the end of T iterations, the algorithm returns

Algorithm 1 AdaBoost(D_n, T)

```
Input: Training set of $n$ examples ($D_n$)
       Number of boosting iterations ($T$)
Output: The classifier ($H$)
Procedure:
 1: $w^1 \leftarrow \left(\frac{1}{n}, ..., \frac{1}{n}\right)$
 2: for $t \leftarrow 1$ to $T$ do
 3:     $h^{(t)} \leftarrow$ LearnWeakClassifier($w^t$)
 4:     $\epsilon_- \leftarrow \sum_{i=1}^{n} w_i^t I\left\{h^{(t)}(x_i) \neq y_i\right\}$
 5:     $\alpha^t \leftarrow \frac{1}{2} \ln\left(\frac{1-\epsilon_-}{\epsilon_-}\right)$
 6:     for $i \leftarrow 1$ to $n$ do
 7:         if $h^{(t)}(x_i) \neq y_i$ then
 8:             $w_i^{t+1} \leftarrow \frac{w_i^t}{2\epsilon_-}$
 9:         else
10:             $w_i^{t+1} \leftarrow \frac{w_i^t}{2(1-\epsilon_-)}$
11:         end if
12:     end for
13: end for
14: return $H = \sum_{t=1}^{T} \alpha^t h^{(t)}$
```

Algorithm 2 AdaBoost.PL($D_{n^1}^1$, ... , $D_{n^M}^M$, T)

```
Input: The training sets of $M$ workers ($D_{n^1}^1, ..., D_{n^M}^M$)
       Number of boosting iterations ($T$)
Output: The classifier ($H$)
Procedure:
 1: for $p \leftarrow 1$ to $M$ do
 2:     $H^p \leftarrow$ AdaBoost$\left(D_{n^p}^p, T\right)$
 3:     $H^{p^*} \leftarrow$ the weak classifiers in $H^p$ sorted w.r.t. $\alpha^{p(t)}$
 4: end for
 5: for $t \leftarrow 1$ to $T$ do
 6:     $h^{(t)} \leftarrow$ Merge$\left(h^{1^*(t)}, ..., h^{M^*(t)}\right)$
 7:     $\alpha^t \leftarrow \frac{1}{M} \sum_{p=1}^{M} \alpha^{p^*(t)}$
 8: end for
 9: return $H = \sum_{t=1}^{T} \alpha^t h^{(t)}$
```

the final classifier H which is a weighted average of all the weak classifiers. The sign of H is used for the final prediction.

Computational Complexity of AdaBoost depends on the weak learner algorithm in line 3. Rest of the operations can be performed in $\Theta(n)$. Lets consider decision stump (decision trees with only two leaf nodes) as weak learners. The cost of finding the best decision stump is $\Theta(dn)$ if the data examples are sorted in each attribute. Sorting all the attributes will take $\Theta(dn \log n)$ time and this has to be done only once before starting the first iteration. So, the overall cost of the T iterations is $\Theta(dn(T + \log n))$.

The pseudocode of AdaBoost.PL is given in Algorithm 2. For a formal description of AdaBoost.PL, let $D_{n^p}^p$ is the data set for the p^{th} worker. The workers compute the ensemble classifier H^p by completing all the T iterations of standard AdaBoost (Algorithm 1) on their respective data sets (line 2). H^p is defined as follows:

$$\{(h^{p(1)}, \alpha^{p(1)}), (h^{p(2)}, \alpha^{p(2)}), ..., (h^{p(T)}, \alpha^{p(T)})\}$$

where $h^{p(t)}$ is the weak classifier of p^{th} worker at t^{th} iteration and $\alpha^{p(T)}$ is the corresponding weight of that weak classifier. The worker then reorders the weak classifiers, $h^{p(t)}$, with increasing order of $\alpha^{p(t)}$ (line 3). This new ordering H^{p^*} is expressed as follows:

$$\{(h^{p^*(1)}, \alpha^{p^*(1)}), (h^{p^*(2)}, \alpha^{p^*(2)}), ..., (h^{p^*(T)}, \alpha^{p^*(T)})\}$$

If, $\alpha^{p(k)} = min\{\alpha^{p(t)} | t \in \{1, 2, ..., T\}\}$ then $\alpha^{p^*(1)} = \alpha^{p(k)}$ and $h^{p^*(1)} = h^{p(k)}$. Now, the reordered $h^{p^*(t)}$s are considered for merging in the rounds of the final classifier. Note that the number of rounds for the final classifier is same as the number of iterations of the workers' internal AdaBoost. But, the t^{th} round of final classifier does not necessarily merges the t^{th} iteration results of the workers. For example, $h^{(t)}$ is formed by merging $\{h^{1^*(t)}, ..., h^{M^*(t)}\}$ (line 6) where, these weak classifiers does not necessarily come from the t^{th} iteration of the workers. This merged classifier, $h^{(t)}$ is a *ternary classifier*, a variant of weak classifier proposed by Schapire and Singer [8] which along with '$+1$' and '-1' might also return '0' as a way of abstaining from answering. It takes a simple majority vote among the worker's weak classifiers:

$$h^{(t)}(x) = \begin{cases} sign\left(\sum_{p=1}^{M} h^p(x)\right) & \text{if } \sum_{p=1}^{M} h^p(x) \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The ternary classifier will answer '0' if equal number of positive and negative predictions are made by the workers' weak classifiers. Otherwise, it will answer the majority prediction. In line 7, the weights of the corresponding classifiers are averaged to get the weight of the ternary classifier. After all the ternary classifiers for T rounds are generated, the algorithm returns their weighted combination as the final classifier.

In a distributed setting, where M workers participates parallelly and the data is distributed evenly among the workers, the computational cost for AdaBoost.PL is $\Theta(\frac{dn}{M} \log \frac{n}{M} + \frac{Tdn}{M})$. The sorting of the T weak classifiers (line 3) will have an additional cost of $\Theta(T \log T)$ time, which becomes a constant term if T is fixed.

III. LogitBoost.PL

LogitBoost [9] is another powerful boosting method that utilizes additive logistic regression model. Unlike AdaBoost, it uses regression functions instead of classifiers; and these functions output real values as prediction. The pseudocode for LogitBoost is described in Algorithm 3. The algorithm maintains a vector of probability estimates

Algorithm 3 LOGITBOOST(D_n, T)

```
Input: Training set of $n$ examples ($D_n$)
       Number of boosting iterations ($T$)
Output: The classifier ($F$)
Procedure:
 1: $F(x) \leftarrow 0$
 2: $p(x_i) = \frac{1}{2}$ for $i = 1, 2, ..., n$.
 3: for $t \leftarrow 1$ to $T$ do
 4:     $z_i \leftarrow \frac{y_i^* - p(x_i)}{p(x_i)(1-p(x_i))}$ for $i = 1, 2, ..., n$.
 5:     $w_i \leftarrow p(x_i)(1 - p(x_i))$ for $i = 1, 2, ..., n$.
 6:     $f_t \leftarrow$ FITFUNCTION$(z, x, w)$
 7:     $F(x_i) \leftarrow F(x_i) + \frac{1}{2} f_t(x_i)$ for $i = 1, 2, ..., n$.
 8:     $p(x) \leftarrow \frac{e^{F(x_i)}}{e^{F(x_i)} + e^{-F(x_i)}}$ for $i = 1, 2, ..., n$.
 9: end for
10: return $F = \sum_{t=1}^{T} f_t$
```

Algorithm 4 LOGITBOOST.PL($D^1_{n^1}$, ... , $D^M_{n^M}$, T)

```
Input: The training sets of $M$ workers ($D^1_{n^1}, ..., D^M_{n^M}$)
       Number of boosting iterations ($T$)
Output: The classifier ($F$)
Procedure:
 1: for $p \leftarrow 1$ to $M$ do
 2:     $H^p \leftarrow$ LOGITBOOST$(D^p_{n^p}, T)$
 3:     $H^{p^*} \leftarrow$ the weak classifiers in $H^p$ sorted w.r.t. their
        unweighted error rate.
 4: end for
 5: for $t \leftarrow 1$ to $T$ do
 6:     $f^{(t)} \leftarrow$ MERGE$\left(f^{1^*(t)}, ..., f^{M^*(t)}\right)$
 7: end for
 8: return $F = \sum_{t=1}^{T} f^{(t)}$
```

(p) for each data point which is initialized to $1/2$ (line 2) and updated during each iteration (line 8). In each iteration LOGITBOOST computes working responses (z) and weights (w) for each data points (line 4,5)[1]. The subroutine FITFUNCTION generates a weighted least-squares regression function of z to x by using the weights w (line 6). The final classifier (F) is a additive model of these real valued regression functions. The final prediction is the sign of F. The cost of finding the best regression function is $\Theta(dn)$ if the data examples are sorted in each attribute. So, the Computational Complexity of LOGITBOOST is $\Theta(dn(T + \log n))$.

In Algorithm 4 we describe our parallel algorithm LOGITBOOST.PL which follows the similar strategy described in Algorithm 2. Like Algorithm 2 LOGITBOOST.PL also distributes the data set among the workers where each worker independently induces their own boosting model. Note that, LOGITBOOST does not assign any weights for the regression functions. Here, we sort the workers' functions with respect to their unweighted error rates:

$$\epsilon = \sum_{i=1}^{n} I\left\{sign\left(f\left(x_i\right)\right) \neq sign\left(y_i\right)\right\} \tag{2}$$

This new reordered functions lists are used to get the merged functions as before. The merged function averages the output of the participating functions:

$$f^{(t)}(x) = \frac{1}{M}\sum_{i=1}^{M} f^{i^*(t)} \tag{3}$$

The final classifier is the addition of all the T merged functions. For fixed T the computational cost for LOGITBOOST.PL is $\Theta(\frac{dn}{M}\log\frac{n}{M} + \frac{Tdn}{M})$ which is same as ADABOOST.PL.

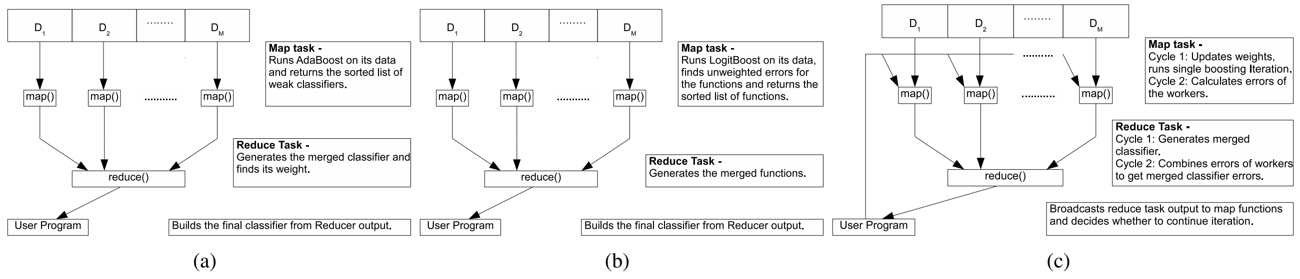
IV. MAP-REDUCE FRAMEWORK

Map-Reduce is a new distributed programming paradigm for cloud computing environment introduced by Dean et al. [6]. *Map-Reduce* has two primary functions: the Map function and the $Reduce$ function. These functions are defined by the user to meet the specific requirements. We used *CGL-MapReduce* [10] for our experiments. It utilizes *NaradaBrokering* [11], a streaming-based content dissemination network, for all the communications. This feature of *CGL-MapReduce* eliminates the overheads associated with communicating via a file system [10].

Figure 1 shows the work flows of ADABOOST.PL, LOGITBOOST.PL and MULTBOOST in *Map-Reduce* framework. Each of the Map functions represents a worker having access to only a subset of the data. For ADABOOST.PL and LOGITBOOST.PL each of the M Map functions runs respective ADABOOST or LOGITBOOST algorithm on their own subset of data to induce the set of weak classifiers (or regression functions). LOGITBOOST.PL has an additional step of calculating the unweighted error rates. Then the base classifiers (or functions) are sorted. These weak classifiers (or functions) along with their weights are transmitted (not applicable for LOGITBOOST.PL) to the $Reduce$ function. After receiving from all the Map functions, the $Reduce$ function merges the weak classifiers (or functions) at the same sorted level and averages (not required for LOGITBOOST.PL) the classifier weights to derive the weights of the merged classifiers. When all the T (total number of boosting iterations) merged classifiers (or functions) are ready, they are sent to the user program and the final classifier is induced. Note that all the boosting iterations are executed in a single burst within the Map function. So, for ADABOOST.PL and LOGITBOOST.PL we need only one cycle of *Map-Reduce* to complete the algorithms.

During each iteration of MULTBOOST [5], each worker builds a weak classifier. These weak classifiers are merged to a single classifier and then the workers measure the weighted errors of this merged classifier on their respective portions

[1]$y^* = (y + 1)/2$, taking values 0, 1.



D1
D2
....
DM
map()
map()
....
map()
reduce()
User Program
Map task - Runs AdaBoost on its data and returns the sorted list of weak classifiers.
Reduce Task - Generates the merged classifier and finds its weight.
Builds the final classifier from Reducer output.
(a)
Map task - Runs LogitBoost on its data, finds unweighted errors for the functions and returns the sorted list of functions.
Reduce Task - Generates the merged functions.
Builds the final classifier from Reducer output.
(b)
Map task - Cycle 1: Updates weights, runs single boosting Iteration. Cycle 2: Calculates errors of the workers.
Reduce Task - Cycle 1: Generates merged classifier. Cycle 2: Combines errors of workers to get merged classifier errors.
Broadcasts reduce task output to map functions and decides whether to continue iteration.
(c)

Figure 1. Map-Reduce work flow for: (a) ADABOOST.PL (b) LOGITBOOST.PL (c) MULTBOOST.

of data. The errors are added to get the total error for the merged classifier and accordingly the workers updates the data points' weights. Then the next iteration begins. So, to complete a single boosting iteration of MULTBOOST the *Map-Reduce* cycle needs to iterate two times.

For the communication cost analysis, let the cost of communications from user program to Map functions, from Map functions to $Reduce$ function and from $Reduce$ function to user program be f, g and h respectively. Then the communication cost for ADABOOST.PL and LOGITBOOST.PL will be $(f + g + h)$. MULTBOOST will take $2T(f+g+h)$ time where T is the number of iterations. Note that, user program communicates with the Map workers parallelly, so the cost does not depend on the number of Map workers.

Table I
THE SYNTHETIC AND REAL-WORLD DATA SETS USED IN OUR EXPERIMENTS.

Datasets	*no. of instances (n)*	*no. of attributes(d)*
yeast	892	8
wineRed	1599	12
wineWhite	4898	12
pendigits	7494	16
spambase	4601	57
musk	6598	167
telescope	19020	11
swsequence	3527	6349
biogrid	4531	5367
isolet	6238	617
d1	200000	20
d2	150000	20
d3	100000	20
d4	5000	2000
d5	5000	1500
d6	5000	1000

V. EXPERIMENTAL RESULTS

In this section, we demonstrate the performance of our proposed algorithms in terms of several performance metrics such as classification accuracy, speedup and scaleup. We compared our results with standard ADABOOST, LOGITBOOST and MULTBOOST in a parallel setup. All our experiments were performed on *Amazon EC2*[2] cloud computing environment. The computing nodes used were of

Table III
COMPARISON OF THE 10-FOLD CROSS-VALIDATION ERROR RATES (STANDARD DEVIATIONS) FOR LOGITBOOST.PL.

Data set	*LogitBoost*	*5 workers LogitBoost.PL*	*10 workers LogitBoost.PL*	*15 workers LogitBoost.PL*	*20 workers LogitBoost.PL*
yeast	0.3408 (0.0500)	0.3577 (0.0562)	0.3610 (0.0435)	**0.3408 (0.0337)**	0.3398 (0.0304)
wineRed	0.2383 (0.0347)	0.2470 (0.0331)	0.2545 (0.0303)	0.2414 (0.0300)	0.2583 (0.0354)
wineWhite	0.2342 (0.0235)	**0.2315 (0.0225)**	0.2344 (0.0210)	**0.2268 (0.0173)**	**0.2327 (0.0173)**
pendigits	0.0657 (0.0113)	**0.0591 (0.0081)**	**0.0602 (0.0094)**	**0.0634 (0.0093)**	**0.0654 (0.0086)**
spambase	0.0550 (0.0080)	**0.0539 (0.0094)**	**0.0541 (0.0069)**	0.0578 (0.0090)	0.0596 (0.0076)
musk	0.0305 (0.0065)	0.0382 (0.0066)	0.0329 (0.0106)	0.0343 (0.0105)	0.0352 (0.0112)
telescope	0.1488 (0.0084)	**0.1434 (0.0077)**	**0.1445 (0.0080)**	**0.1430 (0.0077)**	**0.1457 (0.0083)**
swsequence	0.3296 (0.0215)	0.3243 (0.0229)	0.3314 (0.0216)	0.3307 (0.0258)	0.3342 (0.0221)
biogrid	0.3078 (0.0102)	0.3122 (0.0112)	0.3195 (0.0143)	0.3268 (0.0137)	0.3247 (0.0126)
isolet	0.1347 (0.0086)	**0.1303 (0.0114)**	0.1393 (0.0111)	0.1465 (0.0145)	0.1611 (0.0106)

type *m1.small* configured with 1.7 GHz 2006 *Intel Xeon* processor, 1.7 GB of memory, 160 GB storage and 32 bit *Fedora Core 8*.

We performed several experiments on a wide range of synthetic and real-world data sets. Table I summarizes the 10 publicly available [12] real-world datasets and 6 synthetic data sets used in our experiments. The *spambase* data set is a compilation of user experiences and spam reports about incoming mails. The task of *musk* data set is to learn a model that predicts a new molecule to be either musks or non-musks based on its chemical properties. The *telescope* data set contains scientific information collected by Cherenkov Gamma Telescope to distinguish the two classes: Gamma signals and hadron showers. The *swsequence* data [13] represents the homological function relations that exist between genes belonging to the same functional classes. The problem is to predict whether a gene belongs to a particular functional class (Class 1) or not. The *biogrid* [14] is a protein-protein interaction data that represents the presence or absence of interactions. The *pendigits* data set classifies handwritten digits collected through pressure sensitive writing pad. It was originally designed to be used for multi-class classification with a total of 10 classes (one for each digit from 0 to 9). Instead, we chose to transform it into a binary classification task by assigning the negative class to all even numbers and the positive class to the odd numbers. *Isolet* is a data set from speech recognition domain and the goal is to predict the letter name that was spoken. We also modified this 26 class problem into a binary classification problem by putting first 13 letters in one class and the rest in the other. The biological dataset, *yeast* classifies the cellular localization

[2]http://aws.amazon.com/ec2/

Table II
COMPARISON OF THE 10-FOLD CROSS-VALIDATION ERROR RATES (STANDARD DEVIATIONS) FOR ADABOOST.PL.

		5 workers		*10 workers*		*15 workers*		*20 workers*	
Data set	*AdaBoost*	*AdaBoost.PL*	*MultBoost*	*AdaBoost.PL*	*MultBoost*	*AdaBoost.PL*	*MultBoost*	*AdaBoost.PL*	*MultBoost*
yeast	0.3353 (0.0440)	0.3378 (0.0511)	0.3532 (0.0440)	0.3464 (0.0478)	0.3499 (0.0382)	0.3464 (0.0419)	0.3465 (0.0505)	0.3375 (0.0419)	0.3386 (0.0257)
wineRed	0.2514 (0.0221)	**0.2470 (0.0343)**	0.2658 (0.0300)	**0.2464 (0.0317)**	0.2602 (0.0374)	0.2589 (0.0392)	0.2733 (0.0348)	0.2564 (0.0297)	0.2739 (0.0468)
wineWhite	0.2317 (0.0212)	0.2523 (0.0213)	**0.2425 (0.0147)**	**0.2313 (0.0228)**	0.2542 (0.0140)	0.2346 (0.0232)	0.2536 (0.0181)	**0.2309 (0.0199)**	0.2517 (0.0117)
pendigits	0.0722 (0.0099)	0.0777 (0.0080)	0.0931 (0.0173)	**0.0699 (0.0081)**	0.0878 (0.0136)	**0.0651 (0.0083)**	0.1054 (0.0221)	**0.0642 (0.0102)**	0.1045 (0.0122)
spambase	0.0572 (0.0090)	0.0691 (0.0099)	0.0839 (0.0130)	0.0576 (0.0072)	0.0863 (0.0126)	0.0602 (0.0080)	0.0954 (0.0148)	0.0617 (0.0071)	0.0850 (0.0134)
musk	0.0515 (0.0067)	0.0605 (0.0066)	0.0720 (0.0115)	0.0565 (0.0094)	0.0727 (0.0121)	0.0585 (0.0093)	0.0820 (0.0095)	0.0612 (0.0111)	0.0788 (0.0103)
telescope	0.1551 (0.0063)	0.1668 (0.0116)	0.1675 (0.0054)	0.1599 (0.0086)	0.1694 (0.0119)	0.1659 (0.0104)	0.1727 (0.0112)	0.1595 (0.0132)	0.1685 (0.0100)
swsequence	0.3453 (0.0281)	**0.3289 (0.0263)**	0.3563 (0.0356)	**0.3334 (0.0145)**	0.3698 (0.0288)	**0.3297 (0.0106)**	0.3711 (0.0289)	**0.3368 (0.0174)**	0.3726 (0.0347)
biogrid	0.3136 (0.0050)	0.3229 (0.0521)	0.3556 (0.0671)	0.3180 (0.0134)	0.3625 (0.0356)	0.3447 (0.0401)	0.3613 (0.0412)	0.3291 (0.0511)	0.3746 (0.0421)
isolet	0.1577 (0.0146)	**0.1577 (0.0160)**	0.1922 (0.0105)	0.1601 (0.0162)	0.2050 (0.0169)	0.1605 (0.0127)	0.2344 (0.0103)	0.1661 (0.0150)	0.2358 (0.0177)

sites of Proteins. It is also a multi-class problem with a total of 10 classes. We retained examples only from the two most frequent classes (*CYT, NUC*). The *wineRed* and *wineWhite* data sets [15] model the wine quality based on some physicochemical tests and enumerates the quality score between 0 and 10. In this case, we assigned the negative class to all scores that are less than or equal to five and the positive class to the rest. For the 6 synthetic data sets (*d1-d6*), we used the synthetic data generator *RDG1* available in WEKA [16] data mining tool. *RDG1* produces data randomly by a decision list consisting of a set of rules.

A. Prediction Performance

Table II and Table III report the 10-fold cross validation error rates on the real-world data sets for ADABOOST.PL and LOGITBOOST.PL respectively. For ADABOOST.PL we compare its generalization capability with MULTBOOST and the standard ADABOOST algorithm. MULTBOOST is a variation of ADABOOST, so we did not compare LOGITBOOST.PL with MULTBOOST. In literature we found no parallelizable version of LOGITBOOST to compare with, so LOGITBOOST.PL is compared with standard LOGITBOOST only.

The experiments for ADABOOST were performed using a single computing node. For ADABOOST.PL and MULTBOOST, the experiments were parallelly distributed on a cluster setup with 5, 10, 15 and 20 computing nodes. During each fold of computation, the training set is distributed equally among the working nodes and the induced model is evaluated on the test set. The final result is the average of the error rates for all the 10 folds. For ADABOOST, the error rates are calculated in a similar manner except that, on a single node there is no need for distributing the training set. For all the algorithms the number of boosting iterations is set to 100. In the exact same setting, LOGITBOOST.PL is compared with standard LOGITBOOST.

From Table II, we observe that ADABOOST.PL (with a single exceptions) always performs better than MULTBOOST. In the case where MULTBOOST beats AdaBoost.PL, the differences of errors for the two methods is very low whereas for most of the data sets our algorithm outperforms MULTBOOST significantly. Furthermore, in some cases (highlighted bold in *AdaBoost.PL* columns) our algorithm performs even better than standard ADABOOST. In all other cases, our results are very close to that of the standard ADABOOST. Similarly, the results for LOGITBOOST.PL (Table III) are also very close to original LOGITBOOST and sometimes even better. Such a small compromise in the prediction performance is tolerable if the speedup in computation is significant which really is the case for our proposed algorithms (shown in the next section).

B. Results on Speedup

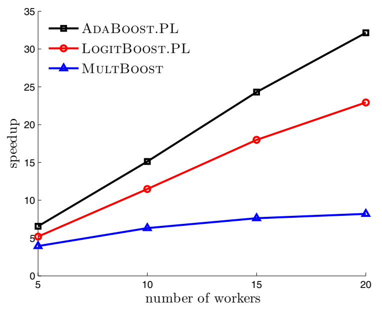
Speedup [17] is defined as the ratio of execution time on a single processor to the execution time for an identical data set on p processors. In a distributed setup, we study the speedup behavior by taking the ratio of baseline (ADABOOST or LOGITBOOST) execution time (T_s) on a single worker to the execution time of the algorithms (T_p) on p workers for the same data set distributed equally. The $Speedup = T_s/T_p$. In our experiments, the values of p are 5, 10, 15 and 20. For the algorithms:

$$Speedup = \Theta\left(\frac{dn\log n + Tdn}{\frac{dn}{M}\log\frac{n}{M} + \frac{Tdn}{M}}\right) \tag{4}$$

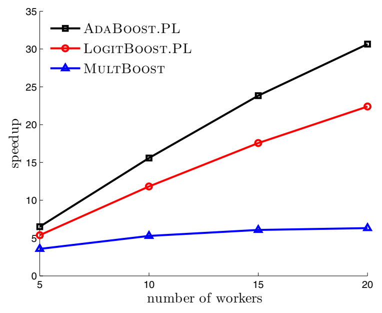
$$= \Theta\left(M\frac{\log n + T}{\log\frac{n}{M} + T}\right) \tag{5}$$

For number of workers, $M > 1$, the inner fraction will be greater than 1. So, we can expect $speedup > M$ for the algorithms.

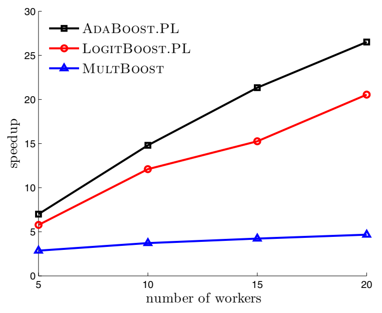
All the algorithms were run 10 times for each data set. We took the ratios of the mean execution times for calculating the speedup. The number of boosting iterations were set to 100. Figure 2 shows the speedup gained by the algorithms on different data sets. From these plots, we observe that, the bigger the data set is the better the speedups are for both of our proposed algorithms, because of the fact that communication cost of the algorithms on smaller data sets tends to dominate the learning cost. For higher number of workers, the data size per workers decreases and so does the computation costs for the workers. But, the communication cost does not change with increasing number of workers. Due to MULTBOOST's higher communication cost, its performance is consistently poor in terms of the speedup.



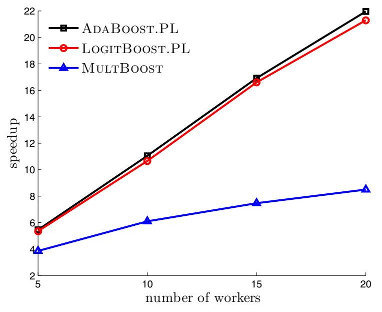
(a) *d1*



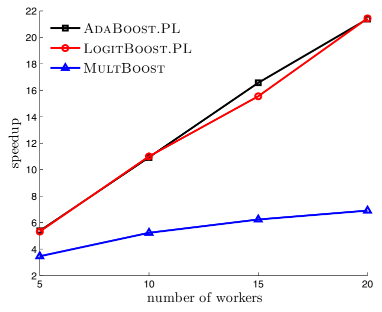
(b) *d2*



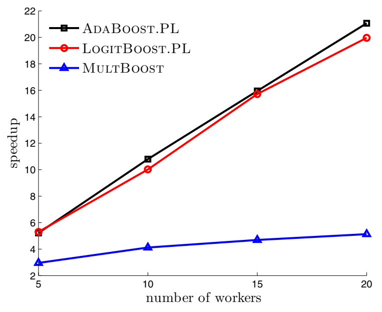
(c) *d3*



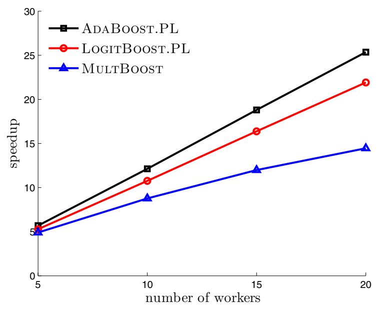
(d) *d4*



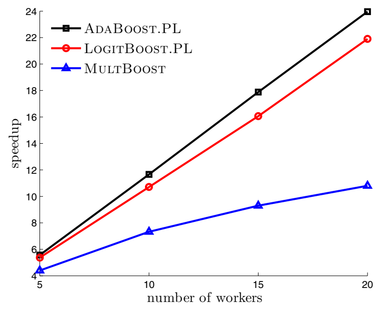
(e) *d5*



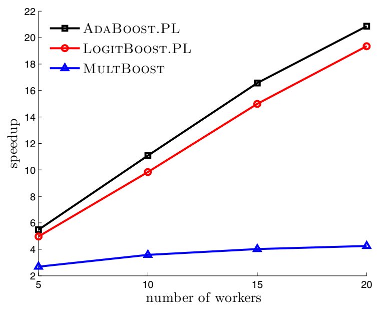
(f) *d6*



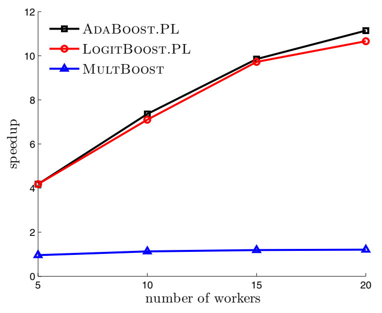
(g) *swsequence*



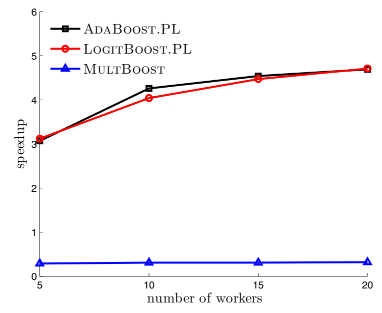
(h) *biogrid*



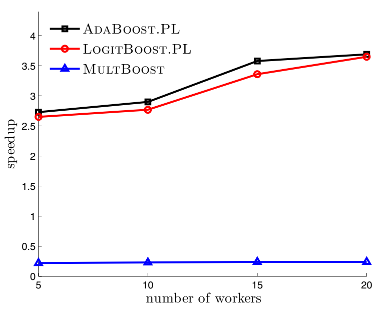
(i) *isolet*



(j) *musk*

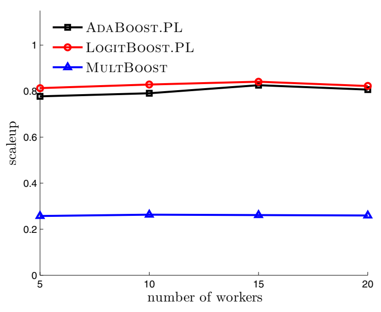


(k) *telescope*

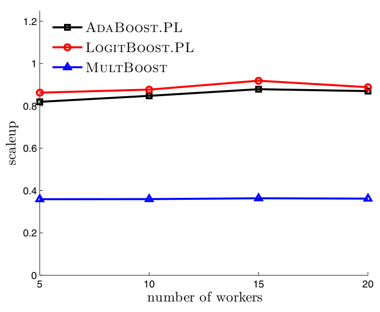


(l) *spambase*

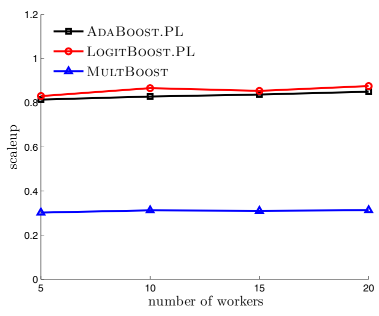
Figure 2. The speedup comparisons for *AdaBoost.PL*, *LogitBoost.PL* and *MultBoost*.



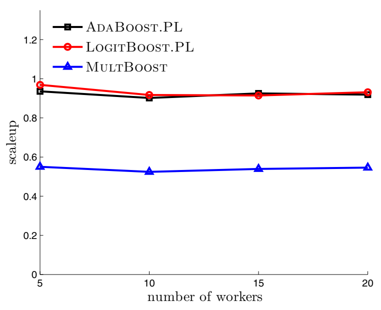
(a) *d1*



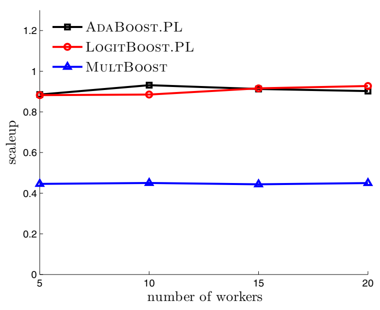
(b) *d4*



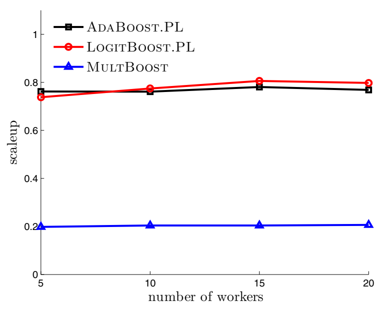
(c) *d5*



(d) *swsequence*



(e) *biogrid*



(f) *isolet*

Figure 3. The scaleup for the AdaBoost.PL, LogitBoost.PL and MultBoost.

C. Results on Scaleup

Scaleup [17] is defined as the ratio of the time taken on a single processor by the problem to the time taken on p processors when the problem size is scaled by p. For a fixed data set, speedup captures the decrease in runtime when we increase the number of available cores. Scaleup is designed to capture the scalability performance of the parallel algorithm to handle large data sets when more cores are available. We study scaleup behavior by keeping the problem size per processor fixed while increasing the number of available processors. For our experiments, we divided each data set into 20 equal splits. A single worker is given one data split and the execution time of baseline (AdaBoost or LogitBoost) for that worker is measured as T_s. Then we distribute p data splits among p workers and the execution time of the parallel algorithm on p workers is measured as T_p. Finally, we calculate scaleup using this equation: $Scaleup = T_s/T_p$. In our experiments, the values of p are 5, 10, 15 and 20. The execution times were measured by averaging 10 individual runs. The number of boosting iterations for all the algorithms were 100.

Figure 3 shows the scaleup of the algorithms for 3 synthetic and for 3 real-world data sets. Ideally, as we increase the problem size, we must be able to increase the number of workers in order to maintain the same runtime. The high and consistent scaleup values for AdaBoost.PL and LogitBoost.PL are evidences of their scalability. Regardless of the increase in the problem size, all that is needed is to increase the available resources and the algorithms will continue to effectively utilize all the workers. Nevertheless, the scaleup behavior of MultBoost is invariably lower.

VI. Related Work

There has been some significant works in accelerating AdaBoost in the literature. These methods essentially gain acceleration by following one of the two approaches: (i) by limiting the number of data points used to train the base learners, or (ii) by cutting the search space by using only a subset of the features. In order to ensure the convergence, both of these approaches increase the number of iterations. However, the net computational time using such methods can be significantly decreased. FilterBoost [18] is a recent algorithm of the former approach which is based on a modification [19] of AdaBoost designed to minimize logistic loss. Following the latter approach for accelerating AdaBoost, Escudero et al. [20] proposed LazyBoost which utilizes several feature selection and ranking methods. Another fast boosting algorithm in this category was proposed by Busa-Fekete et al. [21], which utilizes multiple-armed bandits(MAB) where each arm represents a subset of the base classifier set. However, none of these works described so far explore the idea of accelerating boosting in a parallel or distributed setting.

The strategy of parallelizing of the weak learners instead of parallelizing the boosting itself has seen considerable research efforts. Recently, Wu et al. [22] proposed an ensemble of C4.5 classifiers based on Map-Reduce called MReC4.5. PLANET [23] is a another recently proposed framework for learning classification and regression trees on massive datasets using Map-Reduce. Despite these efforts, there has not been much research to parallelize the boosting itself. Earlier versions of parallelized boosting [24] were primarily designed for tightly coupled shared memory systems. Fan et al. [25] proposed boosting through random samples (r-sampling) or disjoint partitions of the data set (d-sampling). Gambs et al. [5] proposed MULTBOOST algorithm which allows participation of two or more working nodes to construct a boosting classifier in a privacy preserving setting. Though originally designed for preserving privacy of computation, MULTBOOST's algorithmic layout can fit into a parallel setting. In this paper, we already discussed this algorithm in comparison with ours. However, the main problem of these approaches is that they are suited for low latency inter-computer communication environment like: traditional shared memory architecture or single machine multiple processors systems; but not suitable for a distributed cloud environment where usually the communication cost is higher.

VII. Conclusion

We proposed two parallel algorithms for boosting that demonstrate competitive generalization performance with respect to the baselines. We gain significant speedup while building accurate models in a distributed setting. The algorithms also demonstrate very good scalability by efficiently utilizing additional resources for larger sized problems.

References

[1] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *ICML*, 1996, pp. 148–156.

[2] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[3] ——, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[4] D. W. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *J. Artif. Intell. Res. (JAIR)*, vol. 11, pp. 169–198, 1999.

[5] S. Gambs, B. Kégl, and E. Aïmeur, "Privacy-preserving boosting," *Data Min. Knowl. Discov.*, vol. 14, no. 1, pp. 131–170, 2007.

[6] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[7] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997.

[8] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, no. 3, pp. 297–336, 1999.

[9] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *The Annals of Statistics*, vol. 38, no. 2, 2000.

[10] J. Ekanayake, S. Pallickara, and G. Fox, "Mapreduce for data intensive scientific analyses," in *IEEE Fourth International Conference on eScience*, 2008, pp. 277–284.

[11] S. Pallickara and G. Fox, "Naradabrokering: A distributed middleware framework and architecture for enabling durable peer-to-peer grids," in *Middleware*, 2003, pp. 41–61.

[12] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml

[13] M. S. Waterman and T. F. Smith, "Identification of common molecular subsequences." *J. Mol. Biol.*, vol. 147, pp. 195–197, 1981.

[14] C. Stark, B. J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers, "Biogrid: a general repository for interaction datasets," *Nucl. Acids Res.*, vol. 34, pp. 535–539, 2006.

[15] P. Cortez, J. Teixeira, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Using data mining for wine quality assessment," pp. 66–79, 2009.

[16] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[17] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing. Addison-Wesley*, 2003.

[18] J. K. Bradley and R. E. Schapire, "Filterboost: Regression and classification on large datasets," in *NIPS*, 2007.

[19] M. Collins, R. E. Schapire, and Y. Singer, "Logistic regression, adaboost and bregman distances," *Machine Learning*, vol. 48, no. 1-3, pp. 253–285, 2002.

[20] G. Escudero, L. Màrquez, and G. Rigau, "Boosting applied toe word sense disambiguation," in *ECML*, 2000, pp. 129–141.

[21] R. Busa-Fekete and B. Kégl, "Bandit-aided boosting," in *Proceedings of 2nd NIPS Workshop on Optimization for Machine Learning*, 2009.

[22] G. Wu, H. Li, X. Hu, Y. Bi, J. Zhang, and X. Wu, "Mrec4.5: C4.5 ensemble classification with map-reduce," in *ChinaGrid, Annual Conference*, 2009, pp. 249–255.

[23] B. Panda, J. Herbach, S. Basu, and R. J. Bayardo, "Planet: Massively parallel learning of tree ensembles with mapreduce," *PVLDB*, vol. 2, no. 2, pp. 1426–1437, 2009.

[24] A. Lazarevic and Z. Obradovic, "Boosting algorithms for parallel and distributed learning," *Distributed and Parallel Databases*, vol. 11, no. 2, pp. 203–229, 2002.

[25] W. Fan, S. J. Stolfo, and J. Zhang, "The application of adaboost for distributed, scalable and on-line learning," in *KDD*, 1999, pp. 362–366.