Statistics and Its Interface Volume 6 (2013) 585–593

Distributed iteratively reweighted least squares and applications

Colin Chen*

The iteratively reweighted least squares (IRLS) method has been one of the most used methods in statistics estimation. From maximum likelihood estimation for various models, to general estimating equations with longitudinal data, to robust regression, and to general nonlinear parameter estimation, IRLS has been popularly used to find solutions. However, for very large data, the iterative style could be computationally expensive. We propose a distributed version of IRLS, which can be used on a cluster of computers/threads networked with high speed communications. We explore applications of the distributed IRLS on various estimation problems. Using message passing interface (MPI), we implemented the distributed IRLS for robust regression on a cluster with 41 threads. Experiments from the implementation show that the distributed IRLS can solve very large problems more efficiently and economically compared to the classical non-distributed IRLS.

AMS 2000 subject classifications: 60K35.
Keywords and phrases: Iteratively reweighted least squares, Maximum likelihood estimation, Message passing interface, Quasi-likelihood estimation, Robust regression.

1. INTRODUCTION

Most statistical estimation problems can be ultimately reduced to an optimization problem. In regression, assuming y_i, $i = 1, \ldots, n$ are independent observations from the response variable Y and x_{ij}, $i = 1, \ldots, n$, $j = 1, \ldots, p$ are the corresponding covariates. The optimization problem with respect to a p-vector of parameters β for typical regression has the form

$$\min_{\beta} \sum_{i=1}^{n} \rho(x_i, y_i, \beta), \tag{1}$$

where $x_i = (x_{i1}, \ldots, x_{ip})^T$ is the ith covariate vector, ρ is the object function. When function ρ has continuous second derivatives in β, the classical Newton-Raphson method may be used to efficiently solve the optimal solution of β. However, it is quite common that ρ does not have second derivatives in β, for example, Tukey's bisquare function used in robust regression. In such cases, the iteratively reweighted least squares (IRLS) method by Beaton and Tukey (1974, [1]), which uses an alternative explicitly defined weight rather than the Hessian has proven efficient (Holland and Welsch, 1977, [4]).

IRLS is not just used in robust regression. It has a wide range of applications in maximum likelihood estimation with generalized linear models and accelerated failure models with censored data, quasi-likelihood estimation with general estimating equations, estimation of generalized method of moments, and general nonlinear parameter estimation. In the expectation-maximization framework with mixed models, IRLS is also often used for the maximization.

The excellent paper of Green (1984, [3]) with discussions gave a deep investigation of IRLS, from its relation to Newton-Raphson and Fisher scoring to its extensive applications in generalized linear models, linear regression, and nonlinear models. No doubt, after another quarter of century practice in various areas, IRLS roots deeper in scientific computing as well as faces some new challenges. One challenge is with its performance for very large data sets. With nowadays enormous data flow, the iterative style of IRLS could be computationally expensive. In some cases, it could be even slower than the well-known computationally demanding linear programming (Chen, He, and Wei, 2008, [2]).

Multiple machines connected with high speed network have demonstrated powerful and economical ways for extensive computing (Kontoghiorghes, 2006, [7]). We propose a novel distributed version of IRLS on a cluster of computers/threads connected with high speed communication tools, for example, the popular message passing interface (MPI). The main idea of the distributed IRLS is decomposing the weighted least squares, evenly distributing the computing tasks and data on the cluster, and efficiently collecting intermediate results required by the iterative procedure using a $\log_2$ scheme. This paper focuses on efficiently forming the iterative reweighted least squares equations on a distributed system. Solving the equations takes a much smaller portion of the computing time of the IRLS procedure for most large scale applications.

We explore applications of the distributed IRLS on various cases mentioned early. While in some cases, applying the distributed IRLS is straightforward, in other cases either the problem needs to be transformed or the algorithm needs extension.

*Research was initiated when the author was with SAS Institute Inc. Cary, NC, USA.

Section 2 introduces IRLS and its applications in statistics estimation. Section 3 presents the proposed distributed IRLS. Applications of the distributed IRLS in several categories are investigated in Section 4. Section 5 shows some results of an implementation for robust regression. Some extensions and future applications are discussed in Section 6.

2. IRLS

In this section, following the general formulation of Green (1984, [3]), we briefly introduce IRLS, its relation to the Newton-Raphson method and Fisher scoring method, and some applications. More details including the history of IRLS can be found in Green (1984, [3]).

The popularity of IRLS in statistics estimation sprouts from the Newton-Raphson method for maximizing the log-likelihood function L based on a statistical model. The statistical model is often assumed to have a probability (or density) function, from which the log-likelihood function L forms with observed values and predictors.

Let η be the n-vector predictor based on the observed covariant values. It is a function of a p-vector β of parameters of interest (e.g., $X\beta$ in linear regression). p is typically much smaller than n. So the maximization problem is

$$\max_{\beta} L(\eta(\beta)). \tag{2}$$

If $L(\eta(\beta))$ has continuous second derivatives in β, then the Newton-Raphson method solves the likelihood equations

$$\frac{\partial L}{\partial \beta} = 0 \tag{3}$$

by iteratively solving

$$-\frac{\partial^2 L}{\partial \beta \partial \beta^T}(\beta^* - \beta) = D^T \mathbf{u} \tag{4}$$

for an updated estimate β^*, where the $n \times p$ matrix $D = \frac{\partial \eta}{\partial \beta}$, $\mathbf{u} = \frac{\partial L}{\partial \eta}$, and the Hessian $H = \frac{\partial^2 L}{\partial \beta \partial \beta^T}$ are all evaluated at the current value of β. Since the Hessian H depends on randomly observed values, in practice $-H$ may not be positive definite and a solution of β^* may not exist. Green (1984, [3]) provided an approximation,

$$\begin{aligned} -\frac{\partial^2 L}{\partial \beta \partial \beta^T} &\approx \left(\frac{\partial \eta}{\partial \beta}\right)^T E\left(\frac{\partial L}{\partial \eta}\left(\frac{\partial L}{\partial \eta}\right)^T\right)\left(\frac{\partial \eta}{\partial \beta}\right) \\ &= D^T A D, \end{aligned} \tag{5}$$

where $A = E(\frac{\partial L}{\partial \eta}(\frac{\partial L}{\partial \eta})^T)$ is positive definite and (4) becomes

$$D^T A D(\beta^* - \beta) = D^T \mathbf{u}. \tag{6}$$

Assuming D is of full rank p, (6) always has a solution for β^*. The approximation by (5) is called the Fisher scoring method for solving the likelihood equations in (3). (6) represents a standard weighted least squares estimation problem with weight A and design matrix D. The weight matrix depends on β and needs to be updated (reweighted) for each iteration. Thus the exact Newton-Raphson method for solving the likelihood equations in (3) is approximated by iteratively reweighted least squares.

In robust regression,

$$-L(\beta) = \sum_{i=1}^{n} \rho(r_i), \tag{7}$$

where $r_i = y_i - \eta_i(\beta) = y_i - x_i^T\beta$ is the ith residual. As mentioned in the previous section, ρ might not have continuous second derivatives. So, $\frac{\partial^2 L}{\partial \beta \partial \beta^T}$ might not exist. To solve the likelihood equation

$$\frac{\partial L}{\partial \beta} = \sum_{i=1}^{n} \frac{\partial \rho}{\partial r} x_i = 0, \tag{8}$$

let $\psi(r) = \frac{\partial \rho}{\partial r}$ and $w(r) = \frac{\psi(r)}{r}$. Then (8) becomes

$$X^T W X \beta = X^T W Y, \tag{9}$$

where $X = (x_1, \ldots, x_n)^T$, $Y = (y_1, \ldots, y_n)^T$, and W is an $n \times n$ diagonal matrix with diagonal elements $w(r_i)$, which depend on β. With an initial value β, the likelihood equation (8) can be solved iteratively by solving

$$X^T W(\beta) X \beta^* = X^T W(\beta) Y, \tag{10}$$

which are weighted least squares equations with weight $W(\beta)$ and design matrix X. Different from the Newton-Raphson method, IRLS presented in (10) derives directly from the likelihood equations (8) through an explicitly defined weight function $w(r)$, which has a clear statistical interpretation in robust regression (Holland and Welsh, 1977, [4]).

The derivation of IRLS does not necessarily limit to the above two methods. In general, any iterative method with weighted least squares equations, such as those in (6) or (10), has an IRLS formulation. Such easy formulations also contribute to IRLS' popularity.

Convergence of IRLS though generally is guaranteed in practice, theoretical proofs may not be easy in some cases. For example, in robust regression, proofs of convergence only are known with strictly increasing $\psi(r)$ (Huber, 1981, [5]). If convergence is achieved, the solution of IRLS is usually a local maxima/minima, which depends on the initial values. To obtain a global solution, multiple initial values should be tried.

3. DISTRIBUTED IRLS

The major computation involved in IRLS is forming and solving the weighted least squares equations in (6) or (10).

For large data, especially data with a large number of observations, forming these equations takes a large portion of the total computing time. In this section, we decompose the computing involved in (6) or (10) into independent pieces and evenly distribute them on a cluster of computers/threads. Intermediate results are efficiently collected using a $\log_2$ scheme through high speed communications on the cluster. Solutions are broadcasted on the cluster to keep the iterative procedure proceeding until convergence.

3.1 Decomposing

When the weight matrix is diagonal as in robust regression, computing X^TWX can be decomposed according to the rows of X,

$$X^TWX = \sum_{i=1}^{k} X_i^T W_i X_i, \tag{11}$$

where $X^T = (X_1^T, \ldots, X_k^T)$ and W_i is the corresponding sub-diagonal matrix to X_i. Data are split as equal as possible. If $\frac{n}{k}$ is not an integer, the last piece X_k has less data. Correspondingly, the right side in (10) is decomposed as

$$X^TWY = \sum_{i=1}^{k} X_i^T W_i Y_i, \tag{12}$$

where $Y^T = (Y_1^T, \ldots, Y_k^T)$.

If W is not diagonal, but block-diagonal as often seen in the longitudinal data analysis, we could decompose X^TWX and X^TWY according to the block structure. Assuming W has a large number of blocks, we would split the data such that W_i, $i = 1, \ldots, k$ are sub-block-diagonal and roughly of equal size. This may need some shuffling with the data blocks.

If the weight matrix W in (10) (or A in (6)) can be diagonalized or block-diagonalized, the weighted least squares equations are still decomposable. Let

$$W = R^TSR, \tag{13}$$

where R is an $n \times n$ constant matrix (does not depend on β), then (10) becomes

$$(RX)^TSRX\beta = (RX)^TSRY. \tag{14}$$

By replacing X and Y with RX and RY, we have decomposable weighted least squares equations.

The weighted least squares equations in (6) (or (10)) are not decomposable if the weight matrix A (or W) is not diagonalizable. One suggestion is to transform the estimation problem to have independent observations, or independent groups of observations.

The decomposition considered in this paper, such as (11) or (12), is row-wise. When the weight matrix A (or W) is not diagonalizable, it is not decomposable row-wise. However, we can still decompose the multiplication of two matrice column-wise—evenly split the columns of D and A correspondingly and distribute the columns of D and A according to the blocks of the D^TA to the workers. The column-wise decomposition allows each worker to compute one final block of D^TA.

To decompose the multiplication of three matrices D^TAD (or X^TWX) column-wise, we evenly split the work according to the blocks of D^TAD. Due to symmetry, only the lower triangular of D^TAD is considered. Columns of D are distributed to the workers accordingly. Such column-wise decomposition requires each worker to store and update the weight matrix A (or W). It is only suitable when A is relatively sparse (e.g., a thin-banded) weight matrix.

The column-wise decomposition avoids intermediate results aggregation in the cost of repeatedly distributing the same column to multiple workers. However, when the number of variables is large and data have been distributed to the workers (or workers have independent access to the data), the column-wise decomposition may be preferred. While focusing on row-wise decomposition, we will also show some preliminary results for distributed IRLS with column-wise decomposition.

3.2 Data distribution and intermediate results aggregation

Once the weighted least squares problem is successfully decomposed, we can distribute data and tasks on the cluster. These include X_i, Y_i, and codes to compute W_i, $X_i^TW_iX_i$, and $X_i^TW_iY_i$ ($D_i, \mathbf{u}_i$, if they depend on β). For data distribution, the master may just tell each worker the necessary information of its share of data and let the worker read the data in.

The results $X_i^TW_iX_i$ and $X_i^TW_iY_i$ computed by each worker need to be summed up. We use a $\log_2$ scheme—every two workers aggregate their results on one of them, then this worker continues the aggregation on the next level. The final results are accumulated on the last worker k, who also solves the equations and checks convergence. If convergence is not achieved, worker k broadcasts the solution on the cluster and the iterative procedure proceeds.

Data aggregation and broadcasting are supported by communication tools on the cluster. The most popularly used communication tool on a cluster is the message passing interface (MPI). MPI has been developed since 1993 by researchers from Argonne National Laboratory (Pacheco, 1996, [10]) and has become a standard for message passing distributed computing. MPI provides an extensive set of communication subroutines including point-to-point communication, broadcasting and collective communication. It has many implementations on a variety of distributed computers including massive distributed computers, clusters, and networks of workstations.

Using MPI, through client requests, we set up a cluster of computers/threads. One thread takes the role as the master/root, coordinating data distribution and aggregation as well as communicating with the client. The other threads are called workers/nodes. Our design can also be implemented using other communication tools with similar functions.

3.3 The algorithm

The algorithm for the distributed IRLS is summarized in the following steps.

Step 0A. *Data reading and distribution.* There are multiple options. The master can read the entire data, for example the design matrix X and the response Y in robust regression. Then it distributes the data to the workers according to the decomposition. A potentially more efficient way is that the master coordinates the data distribution by telling the workers where and how much they should read. Alternatively, data could be pre-distributed to the workers according to a natural decomposition.

Step 0B. *Initial values.* The initial values for the weights are set to 1 (i.e., $W = I$, the identity matrix), which corresponds to the ordinary least squares estimator. Alternatively, β is set to some initial values and W (D and $\mathbf{u}$ in (6) if they depend on β) is computed based on the initial β values.

Step 1. *Computing with each worker.* Each worker computes its portion of W, X^TWX, and X^TWY (or A, D^TAD, $D^T\mathbf{u}$ in (6)). Except initial values, W is computed according to the updated solution broadcasted on the cluster by the last worker. This is done by each worker independently by assuming a successful decomposition.

Step 2. *Intermediate results aggregation.* Alternate workers communicate neighbors to aggregate the computed X^TWX and X^TWY (or D^TAD and $D^T\mathbf{u}$) repeatedly in a $\log_2$ scheme as shown in Figure 1. Worker k accumulates the final results.

Step 3. *Solving weighted least squares and checking convergence.* Worker k computes the weighted least squares estimates $\hat{\beta} = (X^TWX)^{-1}X^TWY$ and checks the convergence by comparing relative changes of β in consecutive iterations. If convergence is not achieved, it broadcasts $\hat{\beta}$ to other workers and the iteration goes back to Step 1 until the convergence is achieved or reports that the number of iterations exceeds a limit.

Figure 1 presents the flow chart of the distributed IRLS algorithm.

In the case of a diagonal weight matrix W, the distributed IRLS reduces computing time for W, X^TWX, and X^TWY from $O_{compute}(np^2)$ to $O_{compute}(np^2/k)$ with the cost of additional communication time. However, the $\log_2$ scheme of aggregation limits the communication time in the $\log_2$ scale

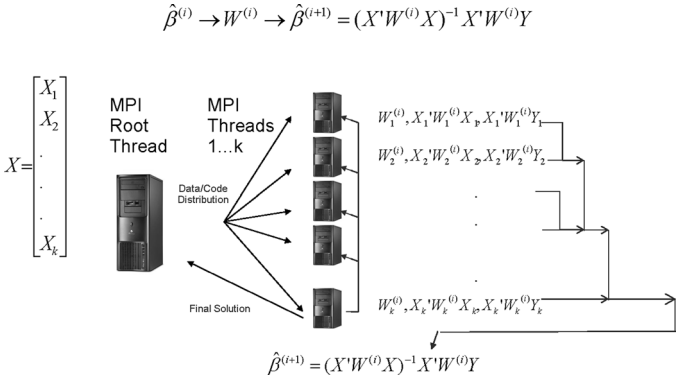

Figure 1. Distributed Iteratively Reweighted Least Squares.

of the number of workers $O_{communicate}(p^2 \log_2 k)$, which allows the algorithm to use a larger optimal number of workers for a large problem compared to classical aggregation methods. The $\log_2$ scheme also solves the communication traffic problem which incurs when all workers try to communicate with the master to pass on intermediate results. As a general implementation rule, we would always like to relieve the master from heavy communication duty with workers, since the master could be busy in communication with clients.

The algorithm focuses on the case of row-wise decomposition. For column-wise decomposition, we need some changes. The main changes are for data distribution and intermediate results aggregation. The master distributes columns of X and Y according to an even split of the lower triangular of X^TWX. To reduce the number of columns to be distributed to the workers, we use some rule of thumb split, which might not be optimal. For intermediate results aggregation, MPI can use the efficient collective communication tool to fill in the entire lower triangular of X^TWX and X^TWY by the last worker.

It should be pointed out that the distributed IRLS presented in this paper is focusing on efficiently forming the normal equations in (6) (or (10)) instead of solving them. The main reasons are that large scale IRLS problems are very often caused by a large number of observations, and that even with a thousand variables, solving those normal equations takes a much smaller portion of the computing time. We do not exclude efficiently using parallel/distributed solvers for these normal equations, e.g., parallel QR or other factorization methods, especially when the number of variables exceeds several thousands.

4. APPLICATIONS

In this section, we apply the distributed IRLS on several popular statistical estimation problems, which include maximum likelihood estimation for independent as well as correlated data, robust regression, and nonlinear parameter models.

4.1 MLE with independent observations

With independent observations, the log-likelihood function $L(\eta)$ has the form

$$L(\eta) = \sum_{i=1}^{n} l_i(\eta_i). \tag{15}$$

So, both the Hessian H and the Fisher information matrix A with respect to $\eta = (\eta_1, \ldots, \eta_n)^T$ are diagonal. Decomposition of the weighted least squares equations in (6) is straightforward.

For generalized linear models in the exponential family with canonical link

$$l_i(\eta_i) = \phi^{-1}(y_i - \eta_i - b(\eta_i)) + c(y_i, \phi), \tag{16}$$

where $b(\cdot)$ and $c(\cdot)$ are some functions and ϕ is a dispersion parameter, $\eta_i = x_i^T\beta$ is the canonical parameter. More generally, Nelder and Wedderburn (1972, [9]) proposed

$$l_i = \phi^{-1}\pi_i(y_i - \theta_i - b(\theta_i)) + c(y_i, \phi), \tag{17}$$

where $b(\cdot)$ and $c(\cdot)$ are prescribed functions, $\{\pi_i\}$ is a set of given weights, and ϕ is the dispersion parameter as in (16). The canonical parameter θ_i is a function of the linear predictor $\eta_i = x_i^T\beta$, $i = 1, \ldots, n$. With this linear predictor, $D = X$ doesn't depend on β.

For accelerated failure models with censored data

$$\begin{aligned} l_i(\eta_i) &= \delta_1 \log \frac{f(\mu_i)}{\sigma} + \delta_2 \log s(\mu_i) + \delta_3 \log F(\mu_i) \\ &\quad + \delta_4 \log(F(\mu_i) - F(\nu_i)), \end{aligned} \tag{18}$$

where $\mu_i = \frac{y_i - x_i^T\beta}{\sigma}$, $\nu_i = \frac{z_i - x_i^T\beta}{\sigma}$. f, s, and F are the corresponding density, survival, and cumulative distribution functions, respectively. δ_1, δ_2, δ_3, δ_4, are indicators for uncensored, right-censored, left-censored, and interval-censored, respectively. So, $\sum_{i-1}^{4} \delta_i = 1$. σ is the scale parameter.

For some distributions with accelerated failure models, the second derivatives of l_i may not be computed explicitly and approximation is often used, for example, the gamma distribution. The quality of such approximation affects the weight in IRLS, and thus the convergence (Meeker and Escobar, 1998, [8]).

The dispersion ϕ or scale σ parameter also needs to be estimated on the cluster. In Section 4.3, we explain how the scale parameter σ in robust regression is estimated on a cluster.

4.2 Quasi-likelihood estimation with correlated data

For longitudinal data, observations within a subject are correlated. If there are n_j observations $y_{j1}, \ldots, y_{jn_j}$ for the jth subject, $j = 1, \ldots, N$, and $n = \sum_{j=1}^{N} n_j$, the quasi-likelihood method of Wedderburn (1974, [12]) assumes the log-quasi-likelihood Q satisfying

$$\frac{\partial Q}{\partial \eta} = V^{-1}(\eta)(Y - \eta) \tag{19}$$

and solves the quasi-likelihood estimating equations

$$\frac{\partial \eta}{\partial \beta}\frac{\partial Q}{\partial \eta} = D^T V^{-1}(\eta)(Y - \eta) = D^T\mathbf{u} = 0, \tag{20}$$

where $V(\eta)$ is the covariance matrix of Y and $\mathbf{u} = V^{-1}(\eta)(Y - \eta)$. Similar arguments as in (5) lead to the weighted least squares for solving β^*

$$D^T V^{-1}(\eta) D(\beta^* - \beta) = D^T\mathbf{u}. \tag{21}$$

Since the covariance matrix is block-diagonal with block sizes n_j, $j = 1, \ldots, N$, the weight matrix $V^{-1}(\eta)$ is also block-diagonal. The decomposition strategy for block-diagonal weight matrix described in Section 3 can be used.

Very often, block sizes n_j, $j = 1, \ldots, N$ are not equal. In practice, we sort the data by subject block sizes in descending order, then group subjects into subgroups such that these subgroups have roughly the same number of rows of the design matrix D. These subgroups are matched to the workers on the cluster.

4.3 Robust regression

IRLS may be the most used method for estimation in robust regression. Adjusting the weights for potential outliers to get resistant estimates provides a direct statistical interpretation for IRLS in robust regression.

The weights derived from (8) are decreasing functions depending solely on the absolute residuals. When outlyingness with the covariates is also considered, the weights should also be functions of the covariates. However, as long as weights w_i can be computed observation-wise, i.e., $w_i = w(r_i, x_i)$, the decomposition in (11) is maintained.

In robust regression, scale equivariance is often required—changing measurement unit should not change the outlyingness. A scale parameter σ is used in the model and the weights are defined as functions of the standardized residuals $\tilde{r}_i = \frac{r_i}{\sigma}$. Often σ needs to be estimated. One method is using the Huber function iteratively

$$(\hat{\sigma}^{(m+1)})^2 = \frac{1}{nh}\sum_{i=1}^{n} \chi_d\left(\frac{r_i}{\hat{\sigma}^{(m)}}\right)(\hat{\sigma}^{(m)})^2, \tag{22}$$

where

$$\chi_d(x) = \begin{cases} x^2/2 & \text{if } |x| < d \\ d^2/2 & \text{otherwise} \end{cases}$$

is the Huber function and d and h are given constants.

Solving σ by (22) on the cluster needs aggregating squares of the residuals. This can be done together with aggregating X^TWX in a $\log_2$ scheme.

Table 1. Time (in millisecond) Used by Each Component of Distributed IRLS per Iteration

Time	Component	Operation
19128	X^TWX	weighted matrix-matrix multiply (25,000 rows)
16761	gather X^TWX	gather and sum results from multiply ($\log_2$)
98	X^TWY	weighted matrix-vector multiply
19	gather X^TWY	gather and sum results from multiply ($\log_2$)
751	$(X^TWX)^{-1}$	factor 1,000 × 1,000 matrix
6	$(X^TWX)^{-1}X^TWY$	matrix-vector multiply (the solution)
1	distribute	broadcast the solution to all workers
90	W	update local potion of W

4.4 Nonlinear models

When the predictor η is a nonlinear function of β, the $n\times p$ matrix, $D = \frac{\partial \eta}{\partial \beta}$ isn't constant and depends on the updated β in each iteration. However, the dependence of D on β does not affect the decomposition in (11) as long as the weight matrix W is diagonal or block-diagonal. The dependence of D on β does add extra computing for updating D for each iteration.

5. IMPLEMENTATION AND NUMERICAL RESULTS

5.1 Distributed robust regression

We implemented the proposed distributed IRLS for robust regression on a cluster of 23 machines, each running 4 opteron 1.8 GHz processors sharing about 16 GB memory. MPI is used for the communication. In the experiment, through client request, we acquired 41 threads spreading on the 23 machines. One thread acts as the master and the rest 40 threads are workers.

For robust regression, we use Tukey's bisquare weight function

$$w(r,c) = \begin{cases} (1-(\frac{r}{c})^2)^2 & \text{if } |r| < c \\ 0 & \text{otherwise} \end{cases}$$

with $c = 4.685$ which corresponds to a 95% asymptotic efficiency for the estimate. For simplicity, we take a fixed scale $\sigma = 1$.

In the experiment, data are simulated with X being a 1,000,000 by 1,000 design matrix and Y being a response vector of length 1,000,000. In our implementation, to further speed up the computing, we have both X and X^T in memory. The total memory occupied by X, X^T, and Y is about 16 GB.

Table 1 shows the time used on each component of the algorithm in a single iteration. The unit of time is a millisecond.

After 6 iterations, the distributed IRLS converges. Each iteration takes nearly the same amount of time. The total computing time is about 4 minutes. It took 5 minutes to distribute X and Y to the 40 workers by the master. In this experiment, only the master has access to the full data.

Table 2. Time (in second) for Distributed Robust Regression with $p = 1{,}000$ and $k = 40$

n	200,000	400,000	600,000	800,000	1,000,000
Time	127	151	178	206	238

We would expect much faster data reading if the workers can read data independently according to the information passed from the master.

For such a large problem, non-distributed IRLS would fail on most commonly used computers. The distributed IRLS achieves a reasonable speed economically.

5.2 Performance

In this subsection, we examine the scalability, accuracy, and complexity of the proposed distributed IRLS.

To demonstrate the scalability, we run the distributed robust regression implemented in the previous subsection with a different number of observations (n), number of variables (p), and number of workers (k). We focus on the scalability of the distributed IRLS with the iterative procedure by excluding the data distribution part, which is not the core of our proposed algorithm. For the iterative procedure, even with the overhead of communications among master and workers, the scalability of the distributed IRLS with n is still largely linear. With p fixed, the time spent on communication is largely constant, while the dominant part of computing of the X^TWX matrix by each worker is propotional to n. Table 2 shows the time (in seconds) for five runs with n = 200,000 to 1,000,000 by 200,000 and p = 1,000 fixed with 40 workers. We specified the convergence rule of relative change of estimates to be 10^{-6}. All five runs stop after 6 iterations. Time in Table 2 is the average of two repeated runs. Figure 2 shows the fitted line of the run time against the number of observations n.

Next, we fix n = 1,000,000 and k = 40 and run the distributed robust regression for p = 200 to 1,000 by 200. Table 3 shows the time of these runs. Each time is the average of two repeated runs. A plot of these times against p^2 shown in Figure 3 suggests that the scalability with p^2 is roughly linear.

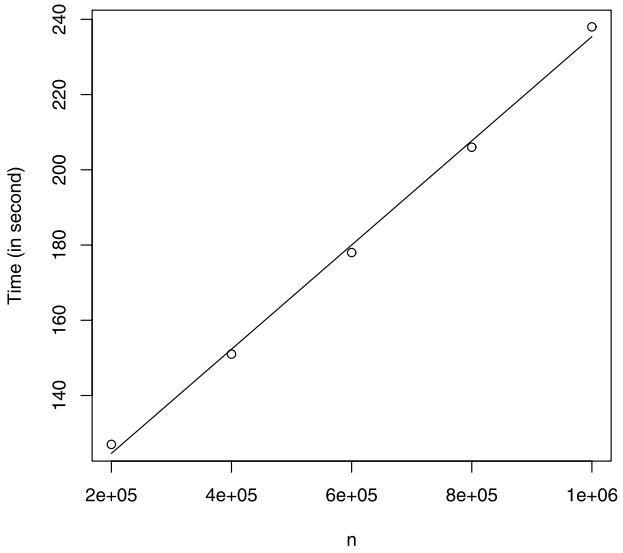

Figure 2. Scalability with n.

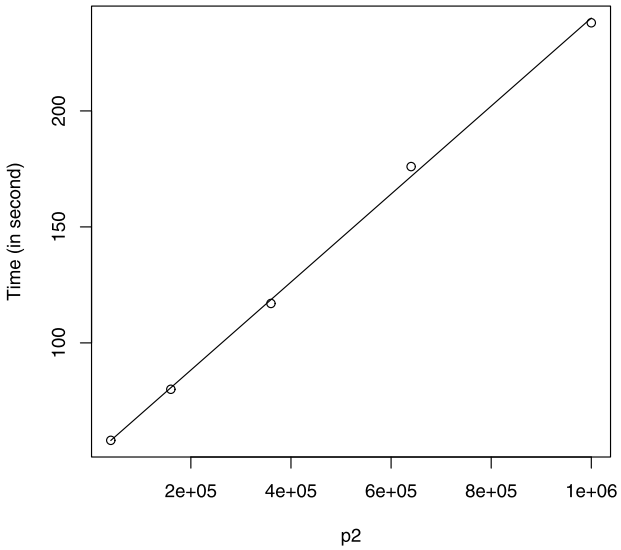

Figure 3. Scalability with p^2.

Table 3. Time (in second) for Distributed Robust Regression with $n = 1{,}000{,}000$ and $k = 40$

p	200	400	600	800	1,000
Time	58	80	117	176	238

Table 4. Time (in second) for Distributed Robust Regression with $n = 1{,}000{,}000$ and $p = 1{,}000$

k	20	40	60	80
Time	355	238	191	164

We also tested the scalability with the number of workers. With $n = 1{,}000{,}000$ and $p = 1{,}000$ we ran the distributed robust regression using three more sets of workers, $k = 20$, 60, and 80. The average run time of two repeats are shown in Table 4. Figure 4 shows the plot of speedup against the number of workers. The solid curve represents the Amdahl's law with a 30% overhead, which fits the four runs well.

To show the efficiency of the distributed system, we tested the large robust regression problem with $n = 1{,}000{,}000$ and $p = 1{,}000$ on a single 2.2 GHz processor sharing 64 GB memory. After three hours, the robust regression was still running its first iteration. We figured out that the processor could not occupy the required memory and disk paging was running. With smaller p, the single processor did run through. We will report this with the credit data in the next subsection.

The accuracy for a large distributed computing system is also important. We examine the accuracy of our proposed distributed IRLS using the relative difference between the estimates from the distributed IRLS and non-distributed IRLS:

Table 5. Relative Difference between Distributed and Non-distributed Robust Regression with $p = 100$

n	200,000	400,000	600,000	800,000	1,000,000
RD	$1.32\ 10^{-7}$	$3.62\ 10^{-7}$	$8.23\ 10^{-8}$	$1.54\ 10^{-8}$	$2.45\ 10^{-8}$

$$\frac{\|\hat{b}_d - \hat{b}_s\|_2}{\|\hat{b}_s\|_2}, \tag{23}$$

where $\hat{b}_d$ is the solution from the distributed robust regression and $\hat{b}_s$ is the solution from a single processor. We ran robust regression with a smaller $p = 100$ due to the long computing time with the single processor and $n = 200{,}000$ to 1,000,000 by 200,000. Table 5 shows the relative difference between the two sets of solutions. These relative differences show that distributed robust regression converges nearly to the same solutions as from the non-distributed version with respect to the convergence criterion 10^{-6} used for both sets of runs.

It should be pointed out that the good accuracy achieved here is partly due to the existence of a unique solution with our robust regression example. If the solution is sensitive by itself, such accuracy may not be achieved. However, data transition with MPI is highly accurate. Poor accuracy is mostly due to the instability of the solutions rather than the distributed system.

The worst case for our proposed distributed IRLS is when the weight matrix W is not decomposable. If W is dense, it needs $np(n + p)$ multiplication operations from $X^T W X$. The iterative reweighted scheme requires W to be updated in each iteration. An UDU decomposition of W does not help too much to fit in the distributed IRLS with row-wise

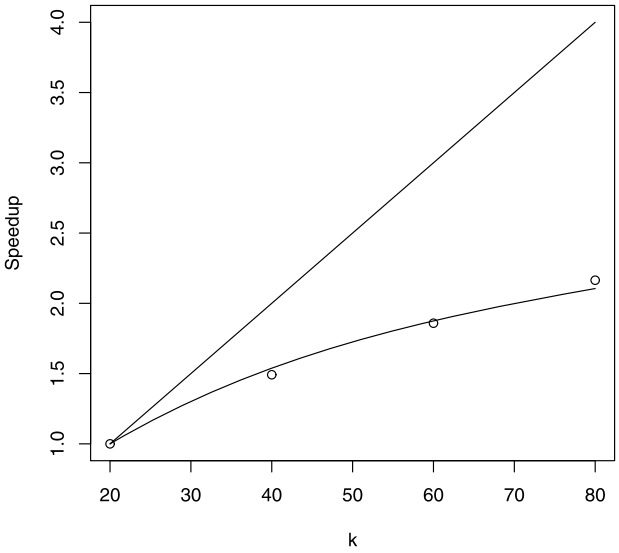

Figure 4. Scalability with k.

Table 6. Time (in second) for Thin-Banded Weight Matrix with Column-Wise Decomposition and $n = 1{,}000{,}000$, $p = 1{,}000$, and $k = 40$

s	1	3	5	7	9
Time	185	283	401	532	677

decomposition. In this case, we implemented the distributed IRLS with column-wise decomposition. As described in Section 3.1, when n is large, only relatively sparse W is suitable to fit in the distributed IRLS. We allow W to be stored and updated by a single worker or to be stored and updated by all workers. With a single worker update, this worker needs to broadcast W to other workers when it finishes the update in each iteration. The balance between the broadcasting and repeated updating by each worker depends on the data size of W. However, our experience recommends the latter.

We tested the distributed IRLS with column-wise decomposition using the previous robust regression example. Although the W matrix is diagonal in this example, we assume it is a banded symmetric matrix with off-diagonal elements being 0. Multiplication between a number and 0 is executed. Table 6 shows the performance (in second) of the implementation. s is the size of the band. $s = 1$ indicates W being simply diagonal.

Table 6 shows that for the diagonal W ($s = 1$) the iterative procedure with column-wise decomposition is faster (took about 3 minutes) than with the row-wise decomposition (took about 4 minutes). This is in the cost of almost twice of the data distribution time (took about 10 minutes).

We would point out that our column-wise decomposition may not be optimal. There is room to improve. Here our goal is to present an initial framework.

5.3 Distributed robust regression with credit data

The credit data include about 16 million credit card accounts. For security, identification information was removed. The variable of interest is the lifetime late fee and interest payments (FI). The explanatory variables were selected characteristics of the accounts (e.g., age of the account), account transaction information (e.g., number of inactive months), and some geographic information. Totally, 102 explanatory variables were selected. After removing records with missing values, the data set has about 14 million observations.

We would like to explore the relationship between FI and the 102 explanatory variables. Due to data quality, we prefer to run robust regression. For such a large data set, the non-distributed version of robust regression is inferior. It took about 17 hours by a single 2.2 GHz processor sharing 64 GB memory on a unix box.

We ran this data set on our distributed grid with 41 threads. It only took about 20 minutes. The difference of the estimates from both the single processor and the distributed grid, as defined in (23), is less than 10^{-6}.

6. DISCUSSIONS

We proposed a distributed version of the iteratively reweighted least squares method based on successfully decomposing the weighted least squares equations and efficiently aggregating intermediate results in a $\log_2$ scheme. As a popular tool for statistics estimation, we explored applications of the proposed method in several categories. Rather than giving implementation details for each application, which is out of the scale of the current paper, we focus on the implementation of the proposed algorithm on robust regression and use it for the exploration of scalability, accuracy, and complexity of the algorithm.

Although related to, but different from the traditional parallel computing, the distributed version of IRLS takes into account the data distribution. There are cases that data are pre-distributed and cannot be collected together due to some security issues (Karr, et al., 2005, [6]). The proposed distributed IRLS fits well in such cases with secured communications. Our distributed IRLS can also be considered as an extension of "embarrassingly parallel" computing implemented by Rossini et al. (2007, [11]) in R, which usually doesn't need intermediate communication among workers.

The current version of distributed IRLS works well for data with large number of observations and the weight matrix is row-wise decomposable. When the weight matrix is not row-wise decomposable and relative sparse, we proposed a version of distributed IRLS with column-wise decomposition. Although there is room to improve, our preliminary results with column-wise decomposition show promise. We are exploring more efficient split methods with the column-wise decomposition. We are also exploring independent data

reading by workers. We have not considered to implement the distributed IRLS on GPUs (graphics processing units), where memory might be a concern.

ACKNOWLEDGMENT

The authors would like to thank Steve Krueger, Chris Bailey, editors, and referees for their helpful comments and discussions.

Received 21 December 2012

REFERENCES

[1] Beaton, A. E. and Tukey, J. W. (1974). The fitting of power series. *Technometrics* **16**, 147–185.

[2] Chen, C., He, X., and Wei, Y. (2008). Lower rank approximation of matrices based on fast and robust alternating regression. *Journal of Computational and Graphical Statistics* **17**, 186–200. MR2424801

[3] Green, P. J. (1984). Iteratively reweighted least squares for maximum likelihood estimation, and some robust and resistant alternatives. *JRSS B* **46**, 149–192. MR0781879

[4] Holland, P. W. and Welsch, R. E. (1977). Robust regression using iteratively reweighted least-squares. *Communication in Statistics, Theory and Methods* **A6**, 813–827.

[5] Huber, P. J. (1981). *Robust Statistics*, John Wiley & Sons, New York. MR0606374

[6] Karr, A. F., Lin, X., Sanil, A. P., and Reiter, J. P. (2005). Secure regression on distributed databases. *Journal of Computational and Graphical Statistics* **14**, 263–279. MR2160813

[7] Kontoghiorghes, E. J. (2006). *Handbook of Parallel Computing and Statistics*, Chapman & Hall/CRC. MR2265409

[8] Meeker, W. Q. and Escobar, L. A. (1998). *Statistical Models for Reliability Data*, John Wiley & Sons, New York.

[9] Nelder, J. A. and Wedderburn, R. W. M. (1972). Generalized linear models. *JRSS A* **135**, 370–384.

[10] Pacheco, P. (1996). *Parallel Programming with MPI*, Morgan Kaufmann, San Francisco, CA.

[11] Rossini, A. J., Luke, T., and Li, N. (2007). Simple parallel statistical computing in R. *Journal of Computational and Graphical Statistics* **16**, 399–420. MR2370947

[12] Wedderburn, R. W. M. (1974). Quasi-likelihood functions, generalized linear models, and the Gauss-Newton method. *Biometrika* **61**, 439–447. MR0375592

Colin Chen
Wells Fargo
Boyds, MD 20841
USA
E-mail address: chennorthc@gmail.com