HPDGLM – Distributed version of glm

User Guide

Contents

HPDGLM – Distributed version of glm

User Guide

1. Introduction

The HPDGLM package provides a distributed Generalized Linear Model. HPDGLM is written based on the distributed R infrastructure developed by HP-Labs. The main functions of the HPDGLM package are:

- dataLoader: Loads a dataset (a set of samples) from a table in HP Vertica to a pair of R darrays.
- hpdglm: A distributed alternative for the R glm function.
- v.hpdglm: Evaluates a model built by hpdglm using the Split-Sample-Validation method.
- cv.hpdglm: Evaluates a model built by hpdglm using the Cross-Validation method.

2. dataLoader

The dataLoader function loads a dataset (a set of samples) from a table in HP Vertica to a pair of R darrays which correspond to responses and predictors of a predictive model. The samples stored in the database (including responses and predictors) must be stored in a single table, and the table must contain a column called *rowid*. The column *rowid* must start from 0, and be contiguous across all values in the column. For example, rowid for a table with five rows would contain the values 0, 1, 2, 3, 4 (see the example in 7.1).

Syntax

```
dataLoader <- function(tableName, resp=list(…), pred=list(…), conf=”RDev”)
```

Arguments:

- tableName: String - The name of the table in the database.
- resp: The list of the name of columns corresponding to responses
- pred: The list of the name of columns corresponding to predictors
- conf: The name of configuration in ODBC.INI file for connecting to the database

Output:

list(Y, X) where

- Y: The darray of responses
- X: The darray of predictors

resp and pred is available as a metadata of Y and X. To retrieve the name of columns of a darray use the command: `colnames(X)` where X can be any darray.

3. hpdglm function

The hpdglm function is a distributed alternative for the R glm function. The R glm function is available in stats package and is available in the standard R distribution. glm is a popular function for performing many forms of regression.

Syntax

```
hpdglm <- function(responses, predictors, family=gaussian, weights=NULL,
na_action="exclude", start=NULL, etastart=NULL, mustart=NULL, offset=NULL,
control=list(...), method="hpdglm.fit.Newton", ...)
```

Arguments

- responses: The darray that contains the vector of responses.
- predictors: The darray that contains the vector of predictors. It may have many columns, but the number of rows and its number of blocks should be the same as responses.
- family: Specifies the family function for regression. The supported family-links are:
  - gaussian(identity)
  - binomial(logit)
  - poisson(log).

  The links (in parenthesis) are the default links for the respective family, so specifying them is optional. The default family is Gaussian.
- weights: An optional darray of ‘prior weights’ to be used in the fitting process. It must be a single column. The number of rows and its number of blocks must be the same as responses. The values must

not be negative (greater than or equal to zero). A zero weight on a sample causes the same to be ignired.

- na_action: Defines the action to take when the data contains missed values. Values of NA, NaN, and Inf in samples are treated as missed values. There are two options for this argument:
  - *exclude:* When *exclude* is selected (the default choice), any the weight of any sample with missed values will become zero, and that sample will be ignored in the fitting process. In the darray which will be created for residuals, the value corresponding to these samples will be NA.
  - *fail*: When *fail* is selected, the function will stop in the case of any missed value in the dataset.
- start: The starting values for coefficients. Optional.
- etastart: The starting values for parameter 'eta' which is used for computing deviance. It must be of type darray. Optional.
- mustart: The starting values for mu 'parameter' which is used for computing deviance. It must be of type darray. Optional.
- offset: An optional darray which can be used to specify an _a priori_ known component to be included in the linear predictor during fitting.
- control: An optional list of controlling arguments. The optional elements of the list and their default values are: epsilon = 1e-8, maxit = 25, trace = TRUE, rigorous = FALSE.
  - epsilon: Used to adjust desired accuracy of the result.
  - maxit: The maximu, number of iterations before achieving the desired accuracy.
  - trace: When this argument is true, intermediate steps of the progress are displayed.
  - rigorous: When this argument is true, some extra checks are performed during fitting procedure. For example, mu and eta may be validating in each iteration to check if the fitted values are outside of the domain. Usually these checks are time consuming; therefore, the default value for this argument is FALSE.
- method: Reserved for the future improvement. The only available fitting method at the moment is “hpdglm.fit.Newton”.

Output

The output is a list of parameters for the learned model:

```
list(coefficients, d.residuals, d.fitted.values, family, d.linear.predictors,
deviance, aic, null.deviance, iter, weights, prior.weights, df.residual, df.null,
converged, boundary, responses, predictors, na_action, call, offset, control,
method)
```

- coefficients: Calculated coefficients
- d.residuals: The working residuals, that is the residuals in the final iteration of the IWLS fit. Since cases with zero weights are omitted, their working residuals are NA. Type darray.
- d.fitted.values: The fitted mean values, obtained by transforming the linear predictors by the inverse of the link function. Type darray.
- family: The family function used for regression.
- d.linear.predictors: The linear fit on link scale. Type darray.
- deviance: Up to a constant, minus twice the maximized log-likelihood.
- aic: A version of Akaike's An Information Criterion, minus twice the maximized log-likelihood plus twice the number of parameters, computed by the aic component of the family.
- null.deviance: The deviance for the null model, comparable with deviance.
- iter: The number of iterations of IWLS used.
- prior.weights: The weights initially supplied. All of its values are 1 if no initial weights used. It is of type darray. The value of weight will become 0 for the samples with invalid data (NA, NaN, Inf).
- weights: The working weights, that is the weights in the final iteration of the IWLS fit. It is of type darray. In order to save memory and execution time, no new darray will be created for weights when the initial weights are all 0 or 1, and it will be simply a reference to prior.weights.
- df.residual: The residual degrees of freedom.
- df.null: The residual degrees of freedom for the null model.
- converged: logical. Was the IWLS algorithm judged to have converged?
- boundary: logical. Is the fitted value on the boundary of the attainable values?
- responses: The darray of responses.
- predictors: The darray of predictors.
- na_action: This item exists only when a few samples are excluded because of missed data. It is a list containing type “exclude” and the number of excluded samples.

- call: The matched call.
- offset: The offset darray used.
- control: The value of the control argument used.
- method: The name of the fitter function used, currently always “hpdglm.fit.Newton”.

4. Peripheral functions

There are a few peripheral functions which can be used in relevance to the output model learned by hpdglm.

4.1. summary

This function prints a summary of the learned model.

4.2. coefficients

This function prints coefficients of the learned model. The abbreviated function is *coef*.

4.3. residuals

This function extracts model residuals in a darray. The abbreviated function is *resid*.

4.4. predict

This function produces predicted values, obtained by evaluating the regression function on provided new data. New data can be either a darray or a normal matrix. Function syntax:

```
predict.hpdglm <- function(object, newdata, type = c("link", "response"), na.action
= na.pass, mask=NULL, trace=TRUE, ...)
```

Arguments:

- object: A built model of type hpdglm.
- newdata: A matrix or a darray containing predictors of new samples.
- type: The type of prediction required. Value can be "link" or "response".

- na.action: A function to determine what should be done with missing values. Currently always na.pass (reserved for future improvement).
- mask: A darray with a single column, and 0 or 1 as the value of its elements. Indicates which samples (rows) should be considered in the calculation.
- trace: When this argument is true, intermediate steps of the progress are displayed.

Output

The output is a matrix or a darray, depending to the type of newdata, which contains predicted values for response.

5. v.hpdglm function

The v,hpdglm function evaluates a model built by hpdglm using the Split-Sample-Validation method. A percent of the data is randomly selected as test data, and a new model is built using the remaining data. Finally, the prediction cost of the new model on the test data is measured.

Syntax:

```
v.hpdglm <- function(responses, predictors, hpdglmfit, cost=hpdCost, percent=30,
sampling_threshold = 1e6)
```

Arguments:

- responses: The darray that contains the vector of responses.
- predictors: The darray that contains the vector of predictors.
- hpdglmfit: A built model of type hpdglm.
- cost: An optional cost function for validation.
- percent: The percent of data which should be set aside for validation.
- sampling_threshold: Threshold for the method of sampling (centralized or distributed). It must be smaller than 1e9. When (`blockSize > sampling_threshold || nSample > 1e9`), the distributed sampling is selected in which a percent of each block is selected for test data. blockSize is the

number of samples in each partition of predictors, and nSample is the total number of samples in predictors.

***Output*:**

```
list(call, percent, delta, seed = seed)
```

- call: The original call to v.hpdglm
- percent: The percent value used at the input
- delta: A vector of length two. The first component is the raw validation estimate of prediction error. The second component is the adjusted validation estimate. Lower cost indicates better fitting. In more detail:
    - 1st-cost = prediction cost of new model on test data
    - 2nd-cost = 1st-cost + (cost of the old model on all data – the cost of the new model on all data)
- seed: The value of .Random.seed when v.hpdglm was called.

6. cv.hpdglm function

This function is implemented for evaluating a model built by hpdglm using the Cross-Validation method. The data is randomly divided into K folds, and the evaluation is repeated K times. Every iteration, one fold is reserved as the test data and the rest is used for training. Finally, the prediction costs of the new models on all folds are aggregated.

Syntax:

```
cv.hpdglm <- function(responses, predictors, hpdglmfit, cost=hpdCost, K=10,
sampling_threshold = 1e6)
```

***Arguments*:**

- responses: The darray that contains the vector of responses.
- predictors: The darray that contains the vector of predictors.
- hpdglmfit: A built model of type hpdglm.
- cost: An optional cost function for validation.
- K: Number of folds in cross validation.

- sampling_threshold: Threshold for the method of sampling (centralized or distributed). It must be smaller than 1e9. When (`blockSize > sampling_threshold || nSample > 1e9 || nSample/K > sampling_threshold`), the distributed sampling is selected in which each block is divided to K folds. blockSize is the number of samples in each partition of predictors, and nSample is the total number of samples in predictors.

***Output*:**

```
list(call, percent, delta, seed = seed)
```

- call: The original call to cv.hpdglm
- K: Number of folds used at the input
- delta: A vector of length two. The first component is the raw validation estimate of prediction error. The second component is the adjusted validation estimate. Lower cost indicates better fitting. In more detail:
    - 1st-cost = the average prediction cost of new models built in K iterations on their test fold
    - 2nd-cost = 1st-cost + (the cost of the old model on all data – the average cost of the new models on all data)
- seed: The value of .Random.seed when cv.hpdglm was called.

7. Examples

7.1. Data Loader

As an example, assume that samples are stored in a table named *mortgage*; the name of response column is *def*, and the names of predictive columns are *mltvspline1*, *mltvspline2*, *agespline1*, *agespline2*, *hpichgspline*, and *ficospline* (see Table 1). Also assume that the name of configuration for the database connection is "*Test*". The command to load all the samples stored in this table is:

```
loadedData <- dataLoader ("mortgage", list("def"), list("mltvspline1",
"mltvspline2", "agespline1", "agespline2", "hpichgspline", "ficospline"),
conf="Test")
```

The darrays for response and predictors are available as `loadedData$Y` and `loadedData$X` respectively. The number of blocks and their size are exactly the same in both darrays. In fact, each row in Y corresponds to the response of a row in X with exactly the same row position. Blocks are row-wise and their number is the same as the number of active R-executors. Please note that the number of rows in the table must be more than the number of active R-executors.

Table 1. Example of a table in the database

rowid	def	mltvspline1	mltvspline2	agespline1	agespline2	hpichgspline	ficospline
1	1	0.760777	0.006632	0.948052	0.906403	0.058021	0.960328
2	0	0.135741	0.205449	0.516031	0.013455	0.827438	0.659125
3	0	0.021796	0.138996	0.862165	0.034211	0.150524	0.345917
4	1	0.271257	0.543280	0.940978	0.891880	0.993050	0.000160
5	1	0.986207	0.053896	0.119611	0.646744	0.819753	0.663289

```
> loadedData <- dataLoader("mortgage", list("def"), list("mltvspline1", "mltvspline2",
"agespline1", "agespline2", "hpichgspline", "ficospline"))
progress: 100%
> names(loadedData)
[1] "Y" "X"
> getpartition(loadedData$X)
      mltvspline1 mltvspline2 agespline1 agespline2 hpichgspline ficospline
 [1,]    0.760777    0.006632   0.948052   0.906403     0.058021   0.960328
 [2,]    0.135741    0.205449   0.516031   0.013455     0.827438   0.659125
 [3,]    0.021796    0.138996   0.862165   0.034211     0.150524   0.345917
 [4,]    0.271257    0.543280   0.940978   0.891880     0.993050   0.000160
 [5,]    0.986207    0.053896   0.119611   0.646744     0.819753   0.663289
```

Figure 1. An example for dataLoader

7.2. Linear Regression

This linear regression example uses a small data-frame called ‘faithful’, which is available in the standard distribution of R (package datasets). This example details how to build a model to predict value of `faithful$eruptions`, given the value of `faithful$waiting`, using four partitions in each darray.

1. Create a darray for response and predictors

   ```
   Y <- as.darray(as.matrix(faithful$eruptions),c(ceiling(length(faithful$eruption)/4),1))
   X <- as.darray(as.matrix(faithful$waiting),c(ceiling(length(faithful$waiting)/4),1))
   ```

2. Build the model

   ```
   myModel <- hpdglm(responses=Y, predictors=X, family= gaussian(link=identity))
   ```

   You can also use the shorthand version:

   ```
   myModel <- hpdglm(Y, X)
   ```

3. Print the summary

```
> summary(myModel)
progress: 100%
progress: 100%
Calculating the covariant matrix
progress: 100%
Building residuals
progress: 100%

Call:
hpdglm(responses = Y, predictors = X)

progress: 100%
progress: 100%
Deviance Residuals:
   Min     Max
-1.299   1.193

Coefficients:
      Estimate Std. Error t value Pr(>|t|)
[1,] -1.874016   0.160143  -11.70   <2e-16 ***
[2,]  0.075628   0.002219   34.09   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.2465251)

    Null deviance: 353.039  on 271  degrees of freedom
Residual deviance:  66.562  on 270  degrees of freedom
AIC: 395.02

Number of Fisher Scoring iterations: 2
```

4. Predict the response for a few new samples:

```
> newSamples <- matrix(c(1:3),,1)
> predict(myModel, newSamples, "link")
          [,1]
[1,] -1.798388
[2,] -1.722760
[3,] -1.647132
```

5. Run Split-Sample-Validation:

```
> testV <- v.hpdglm(Y, X, myModel)
> testV$delta
[1] 0.2281465 0.2274641
```

6. Run Cross-Validation:

```
> testCV <- cv.hpdglm(Y, X, myModel)
> testCV$delta
[1] 0.2445006 0.2442940
```

7.3. Logistic Regression

This logistic regression example uses a small data-frame called ‘mtcars’, which is available in the standard distribution of R (package datasets). This example details how to build a model to predict the value of `mtcars$am`, given the value of `mtcars$wt` and `mtcars$hp` using four partitions in each darray.

1. Create a darray for response and predictors

```
Y <- as.darray(as.matrix(mtcars$am),c(ceiling(length(mtcars$am)/4),1))
X <- as.darray(as.matrix(cbind(mtcars$wt,mtcars$hp)),c(ceiling(length(mtcars$hp)/4),2))
```

2. Build the model

```
myModel <- hpdglm(responses=Y, predictors=X, family= binomial(logit))
```

You can also use the shorthand version:

```
myModel <- hpdglm(Y, X, binomial)
```

3. Print the summary

```
> summary(myModel)
Calculating the covariant matrix
progress: 100%
Building residuals
progress: 100%

Call:
hpdglm(responses = Y, predictors = X, family = binomial)

progress: 100%
progress: 100%
Deviance Residuals:
   Min      Max
-2.254   1.345

Coefficients:
     Estimate Std. Error z value Pr(>|z|)
[1,] 18.86630    7.44356   2.535  0.01126 *
[2,] -8.08348    3.06868  -2.634  0.00843 **
[3,]  0.03626    0.01773   2.044  0.04091 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 43.230  on 31  degrees of freedom
Residual deviance: 10.059  on 29  degrees of freedom
AIC: 16.059

Number of Fisher Scoring iterations: 8
```

4. Predict the response for a few new samples

```
> newSamples <- matrix(c(1:3),3,2)
> predict(myModel, newSamples, "response")
            [,1]
[1,] 0.999979986
[2,] 0.941136088
[3,] 0.005090073
```

5. Run Split-Sample-Validation

```
> testV <- v.hpdglm(Y, X, myModel)
> testV$delta
[1] 0.05269544 0.04091332
```

6. Run Cross-Validation

```
> testCV <- cv.hpdglm(Y, X, myModel)
> testCV$delta
[1] 0.04473004 0.04462759
```

7.4. Poisson Regression

This example for poisson regression uses the same dataset used for logistic regression example.

1. Create a darray for response and predictors

```
Y <- as.darray(as.matrix(mtcars$am),c(ceiling(length(mtcars$am)/4),1))
X <- as.darray(as.matrix(cbind(mtcars$wt,mtcars$hp)),c(ceiling(length(mtcars$hp)/4),2))
```

2. Build the model

```
myModel <- hpdglm(Y, X, poisson)
```

3. Print the summary

```
> summary(myModel)
Calculating the covariant matrix
progress: 100%
Building residuals
progress: 100%

Call:
hpdglm(responses = Y, predictors = X, family = poisson)

progress: 100%
progress: 100%
Deviance Residuals:
    Min      Max
-1.049     1.060
```

```
Coefficients:
      Estimate  Std. Error z value Pr(>|z|)
[1,]  2.287139   0.907534   2.520   0.01173 *
[2,] -1.580433   0.494010  -3.199   0.00138 **
[3,]  0.008831   0.005259   1.679   0.09309 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 23.420  on 31  degrees of freedom
Residual deviance: 10.481  on 29  degrees of freedom
AIC: 42.481

Number of Fisher Scoring iterations: 6
```

4. Predict the response for a few new samples

```
> newSamples <- matrix(c(1:3),3,2)
> predict(myModel, newSamples, "response")
          [,1]
[1,] 2.04528411
[2,] 0.42483043
[3,] 0.08824246
```

5. Run Split-Sample-Validation

```
> testV <- v.hpdglm(Y, X, myModel)
> testV$delta
[1] 0.1481145 0.1661539
```

6. RunCross-Validation

```
> testCV <- cv.hpdglm(Y, X, myModel)
> testCV$delta
[1] 0.14383761 0.09261486
```

7.5. Prediction by Vertica

You can use HPDGLM to connect to HP Vertica to both read data and store results. Assume that you have a table in Vertica database called mortgage. The table has three columns "*def*", "*mltvspline1*", and "*mltvspline2*". You want to find the logistic regression where def is response, and "*mltvspline1*" and "*mltvspline2*" are predictors. First, you make the model in the R environment using the HPDGLM package. Then, you can use the learned coefficients of the model in *vsql* to predict some new samples that are then stored in another table called newSamples. Eventually, you can store the results (the scores of the samples) in a table called prediction.

*** Inside R ***
```
library(distributedR)
library(HPDGLM)
distributedR_start()
```

```
loadedData <- dataLoader("mortgage", list("def"), list("mltvspline1", "mltvspline2"))
myModel <- hpdglm(loadedData$Y, loadedData$X, family=binomial(link=logit),
control=list(trace=FALSE))
coef(myModel)
             [,1]
(Intercept) -2.260290
mltvspline1  3.332633
mltvspline2 -1.537822
```

```
*** Inside vsql ***
CREATE TABLE prediction(p real);
  \set b0  -2.260290
  \set b1   3.332633
  \set b2  -1.537822
INSERT INTO prediction SELECT 1/(1+exp(-(:b0 + mltvspline1 * :b1 + mltvspline2 * :b2))) FROM
newSamples;
```

8. Experiments

The following experiments detail hardware requirements and the scalability of hpdglm function for different numbers of nodes.

Specification of each node for experiments presented in Table 2 and Figure 2

- HP-DL380
- 2 CPU per Node
- CPU model: Intel® Xeon® E5-2670 (20MB Cache, 2.60GHz)
- 120GB RAM
- 10Gb/s Ethernet Network
- CentOS release 6.4 (Final)
- R version 3.0.1

Specification of each node for experiments in Table 3, Figure 3, Figure 4, and Figure 5

- 2 CPU per Node
- CPU model: Intel® Xeon® X5650 (12MB Cache, 2.67GHz)
- 96GB RAM
- 1Gb/s Ethernet Network
- CentOS release 6.4 (Final)
- R version 3.0.1

8.1. Memory requirement

The following table shows the maximum data tested that hpdlgm can handle on a 1 to 8-node cluster. Columns "# node" and "# executors" display the number of nodes and total number of R-executors used in the experiment. Column "Samples" indicates number of samples × number of features of a sample. Column "Input data size" indicated the size of samples in bytes.

Table 2. Experiments to measure memory requirement of logistic regression

# node	# executors	Samples	Input data size	Running time (sec)
1	31	300M×7	16.8GB	145.02
2	62	700M×7	39.2GB	172.375
3	93	1.0B×7	56.0GB	165.996
4	124	1.4B×7	78.4GB	179.925
5	155	1.7B×7	95.2GB	173.465
6	186	2.1B×7	117.6GB	180.064
7	217	2.5B×7	140GB	190.26
8	248	2.9B×7	162.4GB	270.906

8.2. Performance evaluation

While the standard glm function is limited to the resources of a single node, hpdglm utilizes the resources of many nodes. The main benefit of the hpdglm function is achieved when it is employed to process a massive number of samples, because the ratio of communication time to computation time is decreased. In the following experiments, the number of blocks of the input darrays is equal to the total number of instances.

Speedup for logistic regression

To compare the speedup of hpdglm to glm for logistic regression, a set of experiments is performed on fairly small number of samples, 100M×7 (5.6GB). The dataset is selected small enough to be handled by glm. The speedup of hpdglm to glm is illustrated in Figure 2.

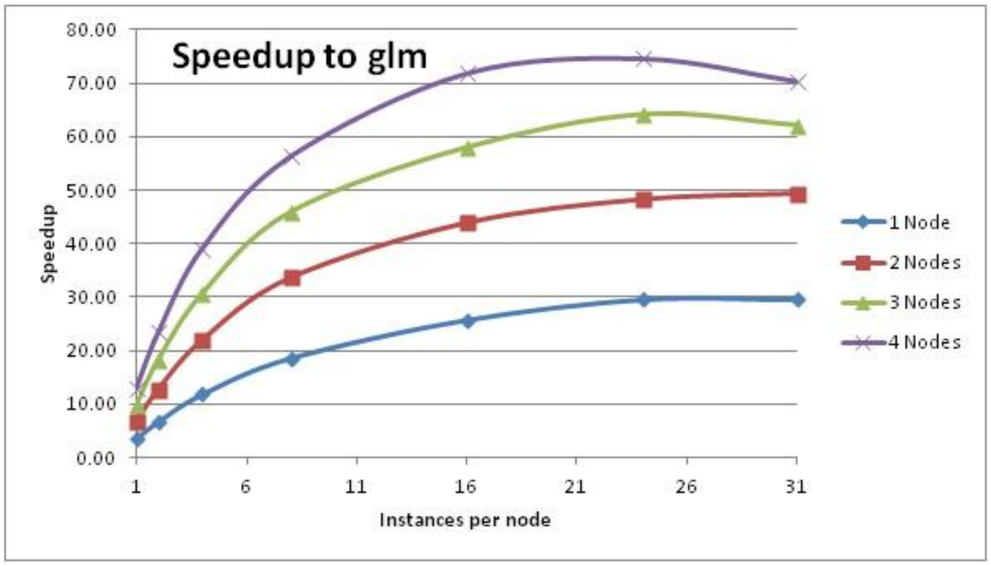

Figure 2. Speedup of hpdglm to glm for logistic regression

Speedup for linear regression

To compare the speedup of hpdglm to glm for linear regression, some experiments are performed on fairly small number of samples, 100M×7 (5.6GB). The dataset is selected small enough to be handled by glm. The speedup of hpdglm to glm is illustrated in Figure 3. Here, the number of instances per node is fixed to 23.

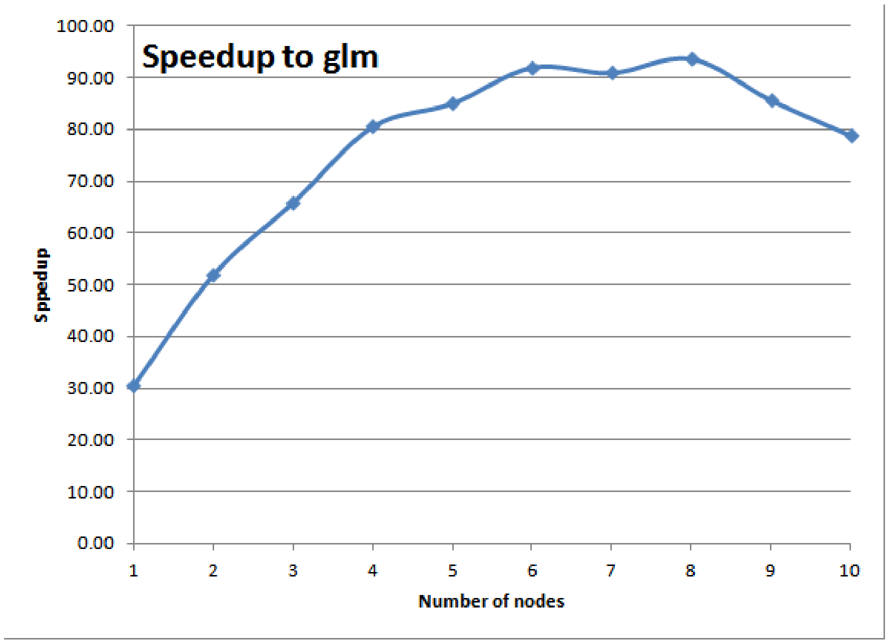

Figure 3. Speedup of hpdglm to glm for linear regression

Strong Scalability for linear regression

In this experiment the strong scalability of hpdglm for linear regression is measured using a dataset of size 750Mx7col (42GB). Here, the number of instances per node is fixed to 23. Figure 4 shows the running time of the experiments. The experiment is not reported for a single node because it was running out of memory.

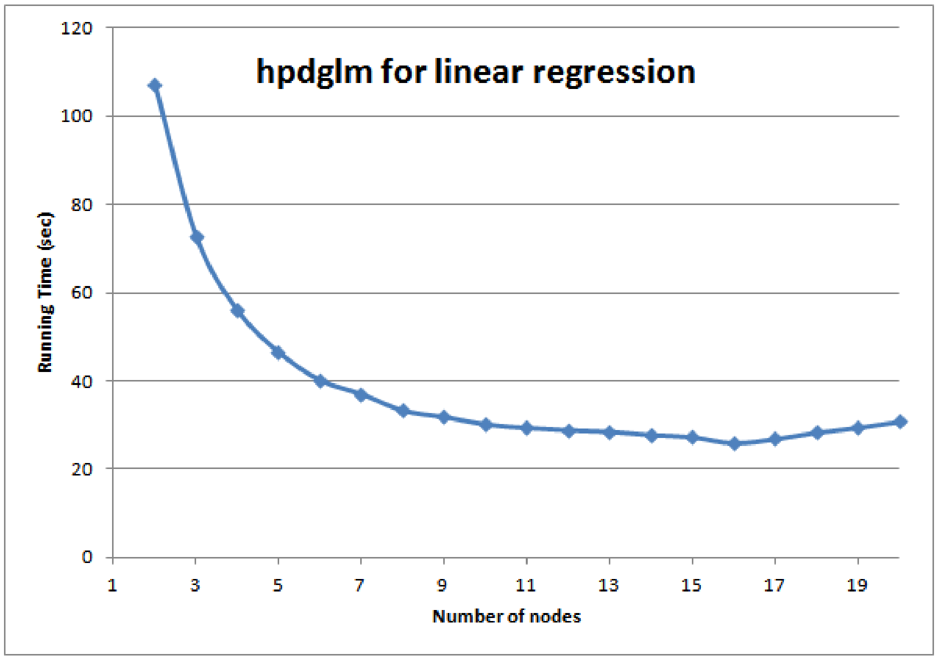

Figure 4. Running time of hpdglm for linear regression

Weak Scalability for linear regression

In this experiment the weak scalability of hpdglm for linear regression is measured. As presented in Table 3, the size of dataset increases along with cluster size (number of nodes). Ideally, the ratio of running times should remain the same. In reality, as it is illustrated in Figure 5 the performance is gradually deteriorating. Here, the number of instances per node is fixed to 23.

Table 3. Running time for measuring weak scalability

Number of nodes	Samples	Size of dataset (GB)	Running time (sec)
1	375Mx7col	21	106.524
2	750Mx7col	42	107.224
3	1125Mx7col	63	107.731
4	1500Mx7col	84	107.388
5	1875Mx7col	105	109.845
6	2250Mx7col	126	109.190
7	2625Mx7col	147	110.598
8	3000Mx7col	168	112.208
9	3375Mx7col	189	111.864
10	3750Mx7col	210	112.412

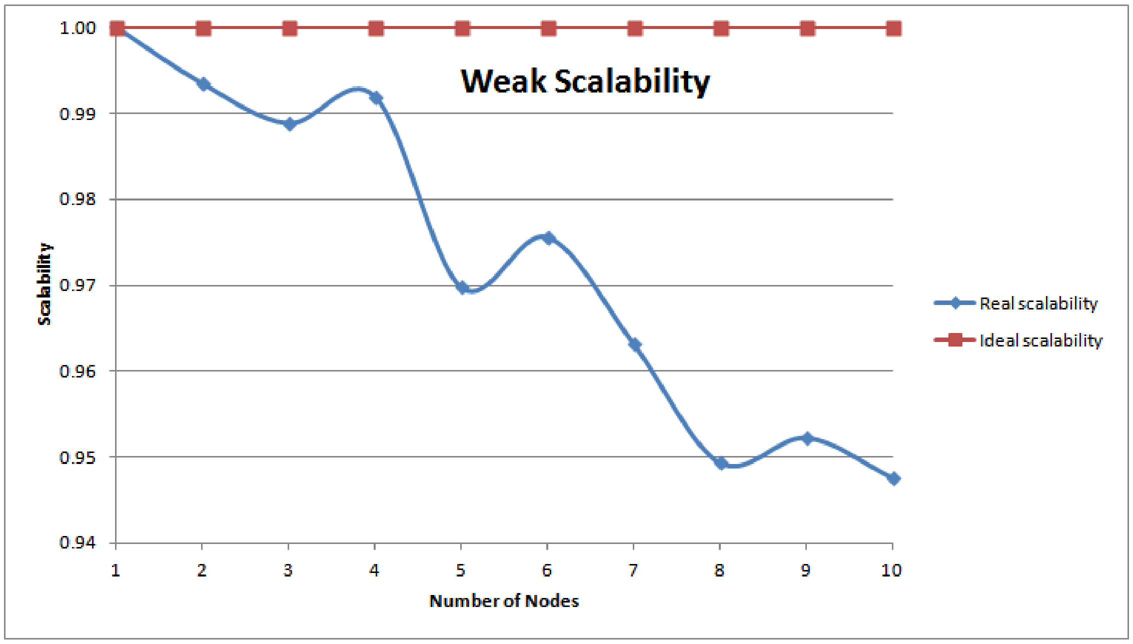

Figure 5. Weak Scalability: size of dataset grows as cluster size grows