PART 1. BATCH COMPUTING MODELS FOR BIG DATA ANALYTICS

1. DISTRIBUTED MODEL FOR SCALABLE BATCH COMPUTING
   -- DISTRIBUTED FILE SYSTEMS

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

Objectives
- Distributed File System
  - GFS2 (Colossus)
- Large scale data analysis using Spark with case study
  - Decision tree/Random Forest

FAQs
- Questions about PA1
  - Send an email to cs535@cs.colostate.edu
- FAQ for PA1 page is available at:
  - http://www.cs.colostate.edu/~cs535/FAQ_PA1.html
- Wikibomb
  - Please see the description of PA1
- No plan for extension
- Difference between original PageRank formula vs. spark example
- Sign-up sheet will be sent out today
- Term Project
  - Google computing cluster credit is available
  - Optional

Reed-Solomon Codes
- Block-based error correcting codes
- Digital communication and storage
- Storage devices (including tape, CD, DVD, barcodes, etc)
- Wireless or mobile communications
- Satellite communications
- Digital TV
- High-speed modems
What does the R-S code do?

- Takes a block of digital data
- Adds extra "redundant" bits
- If an error happens, the R-S decoder processes each block and recovers original data

Reed-Solomon Codes
Quick overview with an example

A Quick Example of the R-S encoding

- 4+2 coding
  - Original files are broken into 4 pieces
  - 2 parity pieces are added
- First piece of data “ABCD”, second piece of data “EFGH”...

Original Data

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
</tr>
</tbody>
</table>

Without 2 rows

<table>
<thead>
<tr>
<th></th>
<th>01</th>
<th>00</th>
<th>00</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td>01</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>1b</td>
<td>1c</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>1c</td>
<td>1b</td>
<td>14</td>
<td>12</td>
</tr>
</tbody>
</table>

A Quick Example of the R-S encoding

- Data loss
  - 2 of 6 rows are lost

Without 2 rows

<table>
<thead>
<tr>
<th></th>
<th>01</th>
<th>00</th>
<th>00</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td>01</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>1b</td>
<td>1c</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>1c</td>
<td>1b</td>
<td>14</td>
<td>12</td>
</tr>
</tbody>
</table>
A Quick Example of the R-S encoding

- Multiplying each side with the inverted matrix

\[
\begin{bmatrix}
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
00 & 00 & 00 & 01 \\
01 & 01 & 12 & 14 \\
1c & 1b & 14 & 12 \\
51 & 52 & 53 & 49 \\
55 & 56 & 57 & 25 \\
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
00 & 00 & 00 & 01 \\
1b & 1c & 12 & 14 \\
11 & 1b & 14 & 12 \\
55 & 56 & 57 & 25 \\
51 & 52 & 53 & 49
\end{bmatrix}
\times
\begin{bmatrix}
A & B & C & D \\
E & F & G & H \\
I & J & K & L \\
M & N & O & P \\
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
00 & 00 & 00 & 01 \\
01 & 01 & 12 & 14 \\
1c & 1b & 14 & 12 \\
51 & 52 & 53 & 49 \\
55 & 56 & 57 & 25 \\
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
00 & 00 & 00 & 01 \\
1b & 1c & 12 & 14 \\
11 & 1b & 14 & 12 \\
55 & 56 & 57 & 25 \\
51 & 52 & 53 & 49
\end{bmatrix}
\]

\[
= \begin{bmatrix}
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
00 & 00 & 00 & 01 \\
01 & 01 & 12 & 14 \\
1c & 1b & 14 & 12 \\
51 & 52 & 53 & 49 \\
55 & 56 & 57 & 25 \\
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
00 & 00 & 00 & 01 \\
1b & 1c & 12 & 14 \\
11 & 1b & 14 & 12 \\
55 & 56 & 57 & 25 \\
51 & 52 & 53 & 49
\end{bmatrix}
\]

A Quick Example of the R-S encoding

- The Inverse Matrix and the Coding Matrix Cancel Out

\[
\begin{bmatrix}
A & B & C & D \\
E & F & G & H \\
I & J & K & L \\
M & N & O & P \\
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
00 & 00 & 00 & 01 \\
01 & 01 & 12 & 14 \\
1c & 1b & 14 & 12 \\
51 & 52 & 53 & 49 \\
55 & 56 & 57 & 25 \\
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
00 & 00 & 00 & 01 \\
1b & 1c & 12 & 14 \\
11 & 1b & 14 & 12 \\
55 & 56 & 57 & 25 \\
51 & 52 & 53 & 49
\end{bmatrix}
\times
\begin{bmatrix}
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
00 & 00 & 00 & 01 \\
01 & 01 & 12 & 14 \\
1c & 1b & 14 & 12 \\
51 & 52 & 53 & 49 \\
55 & 56 & 57 & 25 \\
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
00 & 00 & 00 & 01 \\
1b & 1c & 12 & 14 \\
11 & 1b & 14 & 12 \\
55 & 56 & 57 & 25 \\
51 & 52 & 53 & 49
\end{bmatrix}
\]

A Quick Example of the R-S encoding

- Reconstructing the Original Data

\[
\begin{bmatrix}
A & B & C & D \\
E & F & G & H \\
I & J & K & L \\
M & N & O & P \\
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
00 & 00 & 00 & 01 \\
01 & 01 & 12 & 14 \\
1c & 1b & 14 & 12 \\
51 & 52 & 53 & 49 \\
55 & 56 & 57 & 25 \\
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
00 & 00 & 00 & 01 \\
1b & 1c & 12 & 14 \\
11 & 1b & 14 & 12 \\
55 & 56 & 57 & 25 \\
51 & 52 & 53 & 49
\end{bmatrix}
\times
\begin{bmatrix}
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
00 & 00 & 00 & 01 \\
01 & 01 & 12 & 14 \\
1c & 1b & 14 & 12 \\
51 & 52 & 53 & 49 \\
55 & 56 & 57 & 25 \\
01 & 00 & 00 & 00 \\
00 & 01 & 00 & 00 \\
00 & 00 & 01 & 00 \\
00 & 00 & 00 & 01 \\
1b & 1c & 12 & 14 \\
11 & 1b & 14 & 12 \\
55 & 56 & 57 & 25 \\
51 & 52 & 53 & 49
\end{bmatrix}
\]

Reed-Solomon Codes

Terminology

Properties of Reed-Solomon codes

- RS(n,k) with s-bit symbols
  - Encoder takes k data symbols (blocks) of s bits each
  - Adds parity symbols to make n symbol code word
  - There are n-k parity symbols of s bits each
  - A Reed-Solomon decoder can correct up to t symbols that contain errors in a code word, where 2t = n-k.
  - t = (n-k)/2 for n-k even
  - t = (n-k-1)/2 for n-k odd

Example

- RS(255,223) with 8 bit symbols
  - Each code word contains 255 code word bytes
  - 223 bytes are data and 32 bytes are parity
  - n=255, k=223, s=8, 2t = 32, t=16
  - The decoder can correct any 16 symbol errors in the code word
  - Errors in up to 16 bytes anywhere in the codeword can be automatically corrected.
Large scale data analysis using Spark with case study
Spark software stack

- SQL and DataFrames
- Machine learning
- GraphX
- Spark Streaming

Apache Spark

Regressions and Classifications

- **Regression analysis**
  - A statistical process for estimating the relationships among variables
  - Dependent variable vs. independent variable (‘predictors’)
  - How the typical value of the dependent variable changes when any one of the independent variables is varied, while the other independent variables are held fixed
  - Predicting numerical values
    - Income, temperature, size

- **Classifications**
  - Identifying which of a set of categories a new observation belongs
  - Predicting a label or category
    - Spam, picture of a cat

Vectors and Features

- **Input and output to regression and classification**
- **Example**
  - Predicting tomorrow’s temperature given today’s weather
  - Features (dimensions, predictors, or variables) of today’s weather
  - Today’s high temperature
  - Today’s low temperature
  - Today’s average humidity
  - Whether it’s cloudy, rainy, or clear today
  - The number of weather forecasters predicting a cold snap tomorrow
  - Each of these features can be quantified
    - 13.1, 19.0, 0.73, cloudy, 1

- **Feature vector**
  - These features together, in order
    - (13.1, 19.0, 0.73, cloudy, 1)
    - Features do NOT need to be the same type

- **Categorical features**
  - Features that have no ordering
    - Cloudy, clear

- **Numeric features**
  - Features that can be quantified by a number and have a meaningful ordering
    - 23F, 56F (23F < 56F)
Training Examples

- A learning algorithm needs to train on data in order to make predictions.
- Inputs:
  - Correct outputs (from historical data)
  - “One day, the weather was between 12 and 16 degrees Celsius, with 10% humidity, clear, with no forecast of a cold snap, and the following day the high temperature was 17.2 degrees.”
  - Target (output)
    - 17.2 degree
- Feature vector often includes the target value
  - (13.1, 19.0, 0.73, cloudy, 1, 17.2)

Decision Trees and Forests

- Decision trees can naturally handle both categorical and numeric features
  - Easy to parallelize
  - Robust to outliers
    - A few extreme and possibly erroneous data points may not affect predictions at all
- Random Decision Forests
  - Extended decision tree algorithm

Example: Finding a good pet

<table>
<thead>
<tr>
<th>name</th>
<th>Weight (hg)</th>
<th># of legs</th>
<th>color</th>
<th>Good pet?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fido</td>
<td>20.5</td>
<td>4</td>
<td>Brown</td>
<td>Y</td>
</tr>
<tr>
<td>Mr. Stitch</td>
<td>3.1</td>
<td>0</td>
<td>Green</td>
<td>N</td>
</tr>
<tr>
<td>Nemo</td>
<td>0.2</td>
<td>0</td>
<td>Tan</td>
<td>Y</td>
</tr>
<tr>
<td>Dumbo</td>
<td>1390.8</td>
<td>4</td>
<td>Grey</td>
<td>N</td>
</tr>
<tr>
<td>Kitty</td>
<td>12.1</td>
<td>4</td>
<td>Grey</td>
<td>Y</td>
</tr>
<tr>
<td>Jim</td>
<td>150.9</td>
<td>2</td>
<td>Tan</td>
<td>N</td>
</tr>
<tr>
<td>Millie</td>
<td>0.1</td>
<td>100</td>
<td>Brown</td>
<td>N</td>
</tr>
<tr>
<td>McPigeon</td>
<td>1.0</td>
<td>2</td>
<td>Grey</td>
<td>N</td>
</tr>
<tr>
<td>Spot</td>
<td>10.0</td>
<td>4</td>
<td>Brown</td>
<td>Y</td>
</tr>
</tbody>
</table>

Decision tree for “Finding a good pet” example

Covertype dataset

- Dataset with records of the types of forest covering parcels of land in Colorado, USA
  - Each example contains several features describing
    - Each parcel of land
      - Elevation, slope, distance to water, shade, and soil type
  - The forest cover type is to be predicted
    - From the rest of features
    - Total 54 features
- Used in Kaggle competition
  - It includes categorical and numeric features
  - 581,012 examples
Using `LabeledPoint` for the categorical features

- **One-hot coding**
  - One categorical feature that takes on N distinct values becomes N numeric features, each taking on the value 0 or 1
  - Exactly one of the N values have value 1 and the others are 0
  - Cloudy: 1, 0, 0
  - Rainy: 0, 1, 0
  - Clear: 0, 0, 1

- **1-of-n coding**
  - Cloudy: 1
  - Rainy: 2
  - Clear: 3

Splitting data

- Training, cross-validation, and test
  - 80% of data for **training** and 10% each for **cross-validation** and **test**
  - Training and CV sets are used to choose a good setting of hyperparameters for this data set
  - Test set is used to produce an unbiased evaluation of the expected accuracy of a model built with those hyperparameters

```scala
val Array(trainData, cvData, testData) = data.randomSplit(Array(0.8, 0.1, 0.1))
trainData.cache()
cvData.cache()
testData.cache()
```
Building a DecisionTreeModel on the training set

- Building a DecisionTreeModel on the training set with some default arguments
- Compute some metrics about the resulting model using the CV set

```python
from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.mllib.tree import DecisionTree
from pyspark.rdd import RDD

def get_metrics(model, data):
    predictions_and_labels = data.map(lambda example:
                                       (model.predict(example.features), example.label))
    new_multiclass_metrics = MulticlassMetrics(predictions_and_labels)
    return new_multiclass_metrics

model = DecisionTree.trainClassifier(trainData, 7, Map[Int, Int], "gini", 4, 100)
metrics = get_metrics(model, cvData)
```

Confusion matrix

- 7 x 7 matrix
- The row number corresponds to an actual correct value
- The column number corresponds to a predicted value
- Metrics: confusionMatrix
- Precision: 0.7030630195577938

Precision in multiclass metrics

- Binary classification
  - Positive vs. negative class
  - Precision is the fraction of examples that the classifier marked positive that are actually positive
    - $PPV = \frac{TP}{TP+FP}$
  - Recall is the fraction of all examples that are actually positive that the classifier marked positive
    - $TPR = \frac{TP}{TP+FN}$

- Multiclass problem
  - Positive class vs. negative (all else)

Is 70% accuracy good?

- Classifier that classifies at random in proportion to its prevalence in the training set
- What is the baseline?
  - A broken clock will be correct twice a day
- Randomly guessing a classification would also be occasionally correct

Decision Tree Hyperparameters

- Hyperparameters
  - Values we have to choose by building models
    - Maximum depth, maximum bins, and impurity measure
- Maximum depth
  - Limits the number of levels in the decision tree
  - Useful to avoid overfitting the training data
- Maximum bins
  - feature <= value
  - feature in (value1, value2, ...)
  - A larger number of bins requires more processing time
  - More optimal decision rule
Decision Tree Hyperparameters [2/2]

- Good rule should distinguish examples more meaningfully
- Example
  - A rule that divides the Covtype set into only 1-3 category and 4-7 category would be a great rule
- A good rule divides the training data’s target values into relatively homogeneous or “pure” subsets
  - Minimizing the impurity of the two subsets

Gini impurity

- Measuring impurity degree
- Within a subset, it is the probability that a randomly chosen classification of a randomly chosen example is incorrect
- Includes the sum of products of proportions of classes
  \[ I_d(p) = 1 - \sum_{i=1}^{M} p_i^2 \]
- If the subset contains only one class
  - The value is 0

Entropy

- Borrowed from information theory
- How much uncertainty does the collection of target values in the subset contain?
  \[ I_E(p) = \sum_{i=1}^{M} -p_i \log(p_i) \]

Tuning Decision Trees

- Spark tries a number of combinations of impurity measure, maximum depth or number of bins and reports the results

```
val evaluations = for (impurity <- Array("gini", "entropy"));
  depth <- Array(1, 20);
  bins <- Array(10, 300)) yield {
  val model = DecisionTree.trainClassifier(trainData, 7, Map[Int, Int](), impurity, depth, bins)
  val predictionsAndLabels = cvData.map(example => model.predict(example.features), example.label)
  val accuracy = new MulticlassMetrics(predictionsAndLabels).
  precision ((impurity, depth, bins), accuracy)
  evaluations.sortBy(_._2).reverse.foreach (println) ...
```

Categorical Features Revisited

- Map[Int, Int]()
  - Keys
    - Indices of features in the input Vector
  - Values
    - Distinct value counts
- Empty Map()
  - No features should be treated as categorical
  - All are numeric
- Numeric representation of categorical features
  - It can cause errors
  - The algorithm would be trying to learn from an ordering that has no meaning
Treating the categorical features with one-hot encoding

- Encodes the categorical features as several binary 0/1 values
- Any decision rule on the "numeric" features will choose thresholds between 0 and 1
- All are equivalent since all values are 0 or 1
- Considers the values of the underlying categorical feature individually
- Increases memory usage

Converting one-hot encoding to 1-n encoding

- Specify value count for categorical features 10, 11
- Causes these features to be treated as categorical

Random Forest

- Number of trees to build
  - Here 20
  - "auto" The strategy for choosing which features to evaluate at each level of the tree
  - The random decision forest implementation will NOT even consider every feature as the basis of a decision rule
  - Only a subset of all features
Making predictions

- The results of the DecisionTree and RandomForest training
  - DecisionTreeModel and RandomForestModel objects
- `predict()` method
  - Accepts a Vector object
  - We can classify a new example by converting it to a feature vector in the same way and predicting its target class

```scala
val input = "2709,125,28,67,23,3224,253,207,61,6094,0,29"
val vector = Vectors.dense(input.split(',').map(_.toDouble)
forest.predict(vector)
```