MapReduce Functions on GasDay^{TM} Data Using Hadoop

Darla Ahlert*

02 August 2012

Abstract

The GasDay^{TM} lab at Marquette University forecasts natural gas consumption for 26 Local Distributing Companies around the United States. We have a very large amount of data that has accumulated over the past 19 years, and the lab needs a way to select and process from all of this data to gain insight into our forecasting methods. MapReduce is a pair of functions originally proposed by Jeffrey Dean and Sanjay Ghemawat of Google that allows users to process very large data sets. Hadoop is a software framework created by Doug Cutting in 2004 that allows scientists to distribute vast amounts of data across a cluster of many Linux machines and run MapReduce over the entire data set quickly. The quick turn-around facilitates processing of "Big Data." As examples of MapReduce processing, we found the most accurate type of model, most accurate forecast number, forecast accuracy over time, and the total amount of gas forecasted each year. This is important because we are able to explore different High Performance Computing solutions for anyone with "Big Data." This allows GasDay to improve our forecasting models and grow continuously as a business.

*This work was supported by the National Science Foundation under grant CNS-1063041.

I spent my Summer of 2012 at a Research Experience for Undergraduates (REU) in Marquette University's GasDay^{TM} lab under the guidance of Dr. George Corliss. My project with the GasDay lab was to explore and prototype high performance computing techniques to gather insight from a very large amount of data. This paper is addressed to my successors.

1 Background of this Research Project

Before we get into how I did the things I did and what details I learned from all of it, it is important to understand the background of where all of this came from. Before coming to the GasDay lab, I had never heard of anything referred to as "big data." After arriving at the lab and having the idea explained to me, I realized "big data" is quite simple and probably more common then I originally thought. In short, you encounter "big data" when you have a very large data set that is difficult to work with, whether storing, analyzing, aggregating, querying, or searching the data [3].

The GasDay lab has been running since June of 1993. Its forecasts run every day for 26 Local Distribution Companies (LDCs) around the United States. By Summer 2012, GasDay had over 600,000 days of GasDay forecasts. Each day has around 145 different data elements associated with it. Now, that is a lot of data. GasDay uses three different types of forecasting models; an artificial neural network (ANN) model, a linear regression (LR) model, and a dynamic post processor (DPP) model. They also keep track of the amount of gas that was actually sent out by each LDC, which is called sendout. These four pieces of information are some of the most important in determining the accuracy of the GasDay forecasts.

For GasDay to grow our business and improve our forecasts, additional insights from all of this data could be helpful. That is where Tom Quinn and Samson Kiware came in. Tom and Samson, members of the GasDay lab, did some research during the Spring 2012 semester. They learned about the Hadoop and MapReduce frameworks for handling large amounts of data. Unable to get as far as they would have liked with the project, they handed it off to me when I arrived here at Marquette University for my Summer REU. Here is what they and I found. I will explain the MapReduce and Hadoop frameworks, how they work so well together, how I used the two to gather some prototypical insight into GasDay's data, and the results I gathered from all of this. I will also discuss where the majority of my information can be found on GasDay's Wiki page.

2 MapReduce

We begin with a general overview of the MapReduce framework. MapReduce is a set of two functions that work together, are easily modified to fit many different situations, and allow processing and generating of large datasets. MapReduce was introduced in 2004 by Jeffrey Dean and Sanjay Ghemawat of Google [1]. The set of functions consists of the function `map()` and the function `reduce()`, both written by the user, which work together to aggregate some type of desired results. The function `map()` takes input and produces a set of intermediate key/value pairs: (key, value). The (key, value) pairs then get sorted based on each key and sent to the function `reduce()`. `Reduce()` takes the given (key, value) pairs, merges the similar keys together, and forms a smaller set of (key, value) pairs [1].

The best way to understand this process is to show you an example. The most simple and commonly used example is the word count example [4]. Suppose we have three text files, each containing one of the following sentences:

```
File 1: This is a sentence.

File 2: This is another sentence.

File 3: This could be a sentence.
```

We can use the MapReduce framework to count how many times each word occurs throughout all three of the files. Giving these files as input, the function `map()` goes through each file one piece of data at a time and produces (key, value) pairs. Each word becomes a key, and the number of times the word occurs is its associate value. Since `map()` looks at each piece individually, it is not aware of the number of times it sees of each word and returns a ‘1’ as the value. Repeated words appear multiple times in `map()`’s result list. The output of this mapper might be:

```
[(This, 1), (is, 1), (a, 1), (sentence, 1), (This, 1), (is, 1), (another, 1),
(sentence, 1), (This, 1), (could, 1), (be, 1), (a, 1), (sentence, 1)]
```

The intermediate results are sent to a sort function, which often is included in either the MapReduce or Hadoop frameworks. The sort function sorts the data lexically based on the key in each pair. In this example, it sorts lexicographically based on each word given as a key. The result of the sort function is:

```
[(a, 1), (a, 1), (another, 1), (be, 1), (could, 1), (is, 1), (is, 1),
(sentence, 1), (sentence, 1), (sentence, 1), (this, 1), (this, 1), (this, 1)]
```

Finally, these sorted results are sent to `reduce()`, which combines all of the same keys (or words in this example) and aggregate the values giving the total amount of times each word appears in the data files. As it goes through the list of pairs, `reduce()` compares the key it is currently considering with the key it previously had examined. If the current key matches the previous key, `reduce()` aggregates the values. In this example, the first and second keys are both ‘a,’ so `reduce()` adds the two values together to yield a new value of ‘2.’ Our final output is:

```
[(a, 2), (another, 1), (be, 1), (could, 1), (is, 2), (sentence, 3), (this, 3)]
```

This is a simple example of how MapReduce works. Although the data given as input may only be three small text files containing one sentence each, this MapReduce pair could run over millions of documents with thousands of sentences in each file [2]. Another great way to understand is to see a pictorial example which is given below.

Figure 1 depicts a “Shape counter” example of a MapReduce program. The shapes are given to `map()`. `Map()` runs and returns its output. The key is each shape, and the value is the

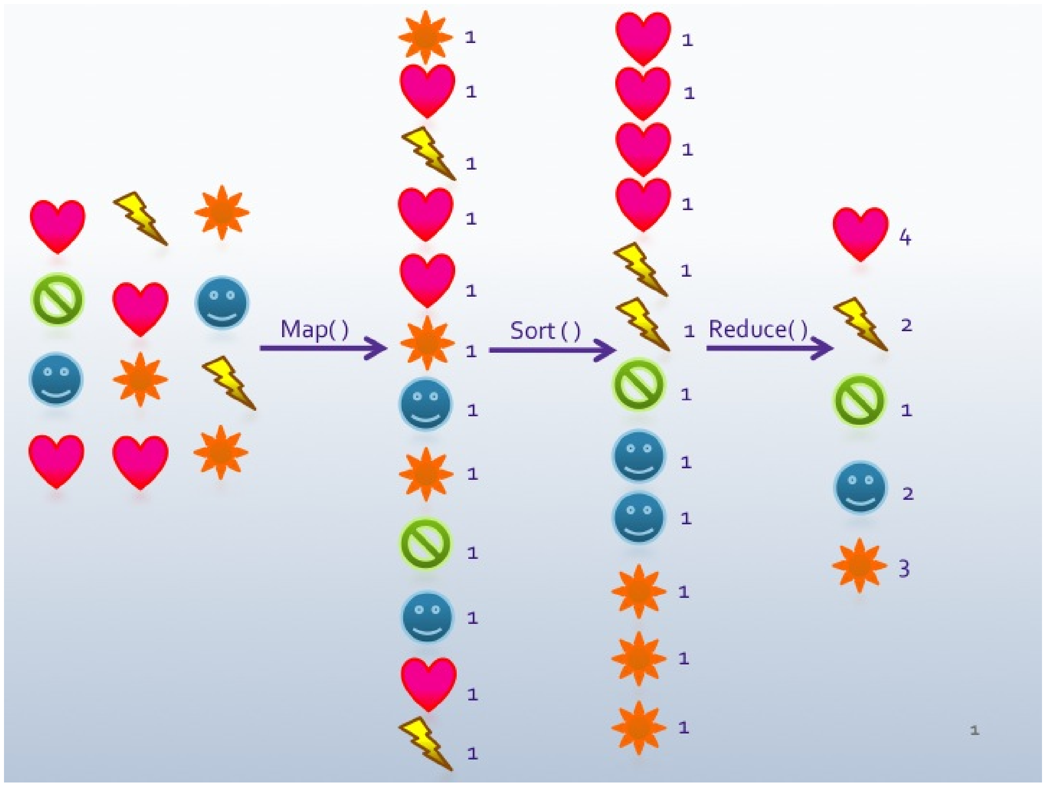


Figure 1: MapReduce Example

number ‘1.’ These (key, value) pairs get sorted. In this example, I sorted them lexically based on a name I gave each shape, such as ‘heart’ or ’lightning bolt.’ The sorted results are sent to `reduce()`, which gives the final (key, value) output.

The MapReduce framework is so powerful because the (key, value) pairs can be more complicated then the ones given in either of these examples, as more examples given later in the paper show. Now that we have a way to process large amounts of data, we need a way to store all of it. That is where the Hadoop framework becomes important, so I will discuss it next.

3 Hadoop

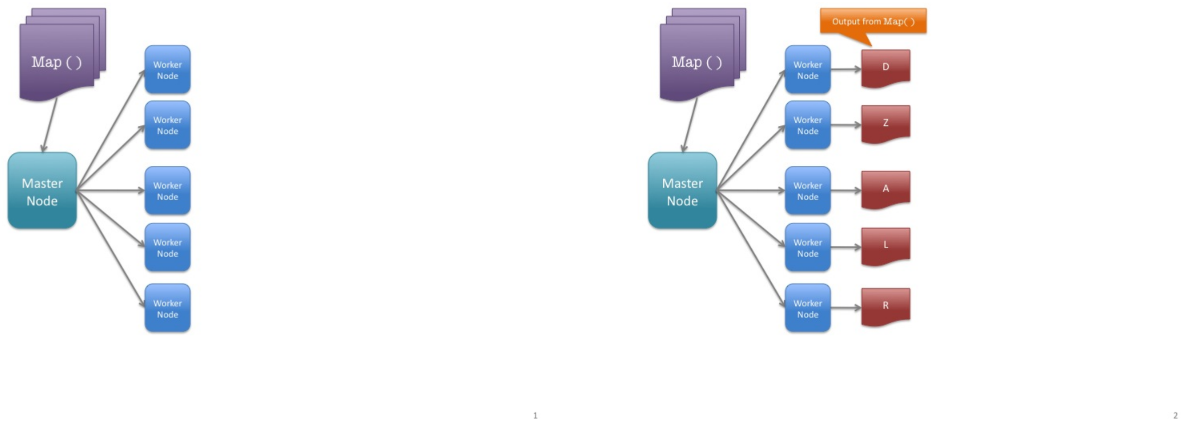
Hadoop is a software framework that allows users to distribute large amounts of data across multiple Linux machines, called a cluster, to store and analyze this data. We can think of Hadoop as an operating system. A normal operating system has many different aspects, including file management, processor management, memory management, and the management of communication between applications. Hadoop has many of these properties, and they are distributed across all of the machines in the cluster [2]. The Hadoop Distributed File System (HDFS) and a Java-based API are provided by Hadoop. HDFS is like an operating system’s file manager because it stores the input data and distributes the data to all of the machines in its cluster when the appropriate command is given to the machine in charge. HDFS manages all of the data that is processed. It splits the input data into multiple pieces that are normally anywhere from 16 to 64 megabytes of data. HDFS uses replication to provide availability and keeps multiple copies of almost everything on the cluster [2]. If at some point one of the machines fails to work, there are other copies of everything on that machine, allowing the programs and processes to continue with little delay. This is like an operating system’s memory manager. Also, using a utility called Hadoop Streaming, we are able to run jobs in computer languages other than Java, such as Python, the language I used to write my MapReduce programs [6].

Michael Noll's tutorials help users set up a Hadoop cluster, and they are what we used for this project [5]. Hadoop clusters are set up so that one of the machines in the cluster is the "master," and the rest of the machines are set up as "workers" or "slaves." The master is similar to an operating system's process manager because it is in charge of the entire cluster and keeps track of all of the important information, such as the state of each node. The master knows which machines are currently idle and which are busy completing tasks [1]. Along with knowing which machines are idle, the master keeps track of machines that are failing to work. The master periodically sends a message to each worker node. If there is no response, the master knows that worker is not running properly [1]. Having all of this information allows the cluster to run with minimal errors in the process. It also allows the Hadoop framework to manage MapReduce tasks.

4 Working together

Now, I can discuss how the Hadoop and MapReduce frameworks work so well together. Hadoop is able to control everything that is happening throughout the cluster with the assigned master node. It keeps track of a large amount of information regarding any MapReduce task and makes sure that each map and reduce task has no dependence on any other task, except for the step in which the mappers feed their output into the reducers. This allows certain tasks to process in parallel. Here is how MapReduce is started.

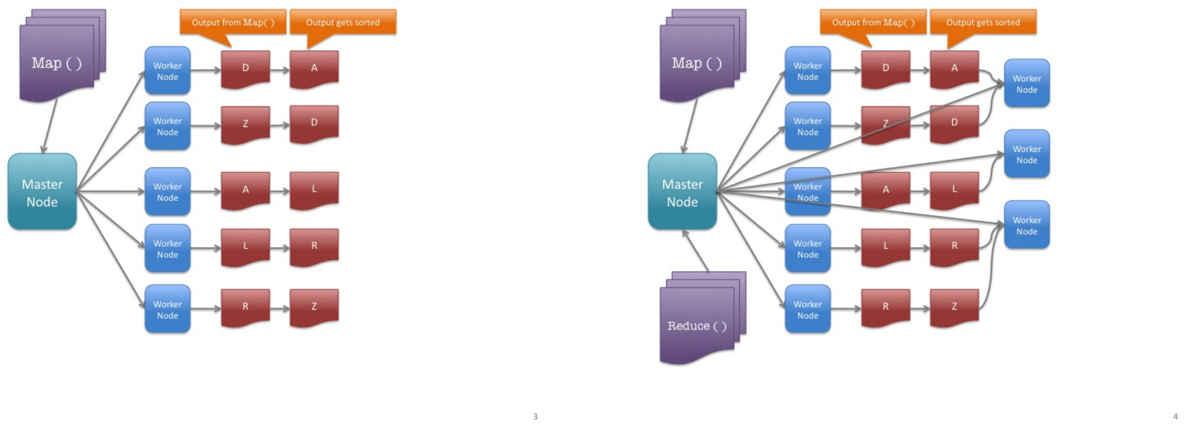
You can find important, detailed information related to this project on GasDay's Wiki page at *www.wiki.gasday.com*, also discussed later in this paper under the "Wiki and Other Resources" section. The user begins by typing a command to start the cluster. Once the cluster is up and running, the user types in another command to start the MapReduce framework. The second command contains pertinent information needed to run the programs, such as the location of the functions `map()` and `reduce()`, the location of the input data in the HDFS, the location in which the user would like the final output file, and any other locations of important files that need to be distributed throughout the cluster [5]. The master takes these files, splits them into the appropriate pieces, and distributes them throughout worker nodes that are marked as idle. The data is split in a particular way which allows each of the machines to run `map()` in parallel with one another [2]. The output of `map()` gets buffered into memory. The locations are stored in the master node, which notifies idle worker nodes that this data is ready for the next step in the process. The workers then read a certain amount of data from the `map()` output and sort this data based on the key given in each output pair, grouping together intermediate keys that are the same. `Reduce()` runs over the sorted key/value pairs and aggregates the results. Just as the input of `map()` is split in a certain way, the input of `reduce()` is also split in a certain way. This allows each of the machines to run `reduce()` tasks in parallel with one another. The results from each `reduce()` are aggregated into one final output which is sent to the location the user specifies in the second command they give [2]. We can see this process in Figures 2 through 4, a six-step diagram showing the MapReduce framework working with Hadoop to gather desired results from the data distributed across the cluster.



Step One Step Two

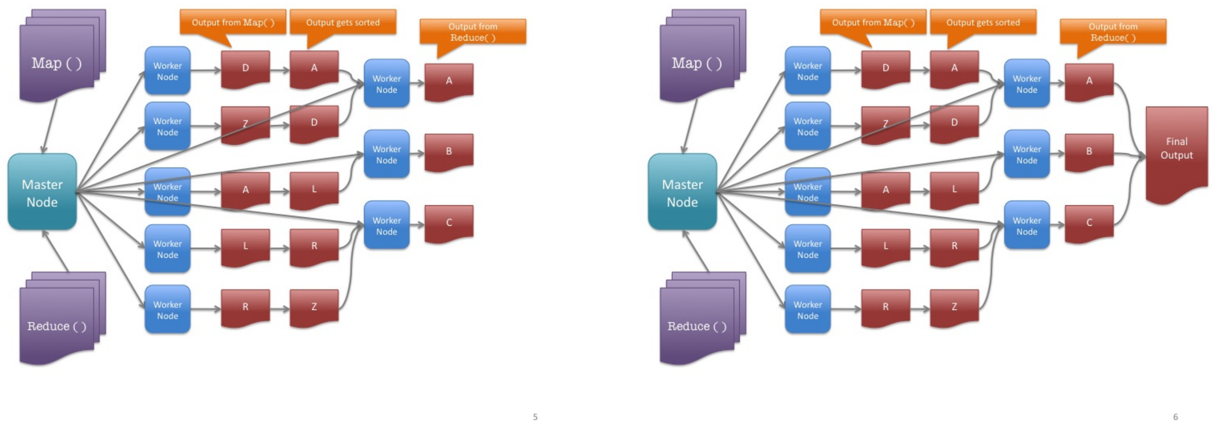
Figure 2: Master sending `map()` and workers running `map()`

Parallelization is key to these two frameworks being successful. Suppose, in my previous



Step Three Step Four

Figure 3: `Map()` output gets sorted and sent to `reduce()`



Step Five Step Six

Figure 4: `Reduce()` output is aggregated into one final output

word count example, we had 500,000 text files with a million words in each file. Running MapReduce over this data using only one computer will work, but it would take a long time. The ability to run MapReduce over this data using multiple machines potentially speeds the process up, but it is necessary to add in the time it takes to split the data up onto each machine and aggregate it back together once we have output from all of the machines. Hadoop is efficient because it does all of this work for us, and we are able to run MapReduce in parallel over the cluster, speeding up the entire process. Now, I can talk about how I used these two frameworks to gather some prototypical results for the GasDay lab.

5 My Work

When I arrived at the beginning of the summer, Tom and Samson showed me all the details needed to get my own cluster running, as well as a couple of example programs they wrote and ran over the data they had gathered. This was enough to get me started. I was given eleven machines to create my own cluster, and I was able to get Hadoop running on each machine as a "single-node" cluster, meaning that each machine was running a Hadoop cluster within itself. The next step was to connect all eleven machines into one cluster. After a few unsuccessful attempts, I was able to get eight machines connected and successfully running as a Hadoop cluster. Eight machines is better than zero, and more machines then Tom and Samson had in their cluster. I took this as a win.

This project is intended to see if the MapReduce and Hadoop frameworks are a good choice for the GasDay lab to analyze all of our data. Since this is not an exhaustive test of GasDay's data, I ran my code on about 575 data files, which contain almost 3.3 million forecasts. There are many more data files associated with GasDay that were not included in my results for a few reasons. GasDay switched database formats during that time. When Tom and Samson started this project, they extracted data using the old database format, and because of this, all of the data was not readily accessible to populate the cluster. Since the intent of my project was to prototype an approach, I did not worry about extracting any more data and used only the data that Tom and Samson originally extracted. This can be seen in the results that I have gathered. The data I have from more recent years does not even begin to cover the amount of data that is actually in the databases. Because of this, some of the graphs look a little strange in more recent years. Therefore, when this project is continued, the rest of the data needs to be extracted from all of the databases to get more complete results.

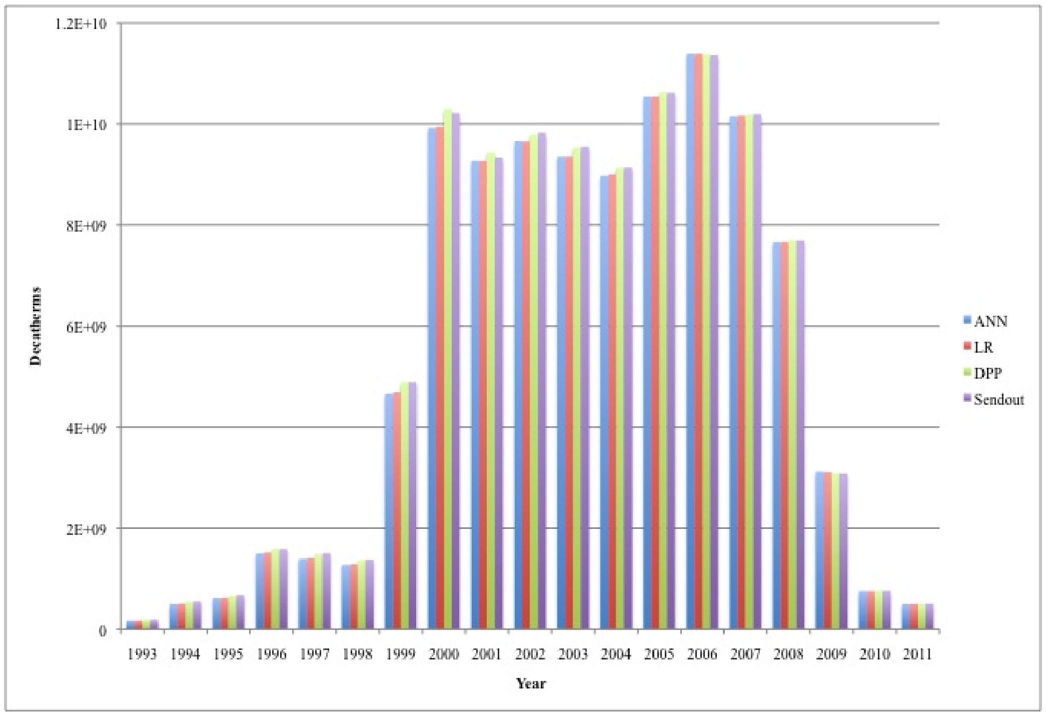


Figure 5: Total amount of gas forecast each year

I was able to complete five MapReduce programs that successfully ran over all of the data on the Hadoop cluster. Using the three forecast models mentioned in the first section (ANN, LR, DPP), as well as the sendout, I was able to gather some prototypical results. Figure 5 shows the total amount of decatherms of gas forecast for each type of model as well as the actual sendout. From the graph we can see that until more recent years, the general flow of the columns is increasing. GasDay has grown vastly over the 19 years we have been running, and we are continuing to grow today.

Figure 6 shows the mean average percent error (MAPE) for all three types of forecasting models. Equation (1) shows how to calculate the MAPE for each model. To find the average percent error for each day, you take the absolute value of the actual sendout subtracted from the forecasted amount, divide the entire quantity by the actual sendout, and then multiple by 100 to get the percent. To calculate this over an entire data set, you take the average of that equation over all the days in the data set, giving you the MAPE.

$$\frac{100}{n} * \frac{\sum_{i=1}^{n} |\text{forecasted amount}_i - \text{actual sendout}_i|}{\sum_{i=1}^{n} \text{actual sendout}_i}\text{, where } n \text{ is the number of days.} \tag{1}$$

The MAPE is used for all of the graphs in this paper except for Figure 5. From Figure 6 we can see our most accurate type of model is the DPP model, whose MAPE is 4.54. The DPP model is a little over one percent better than the other two models, and with the amount of data we have, that is a huge difference. The next best model is the ANN model, with MAPE of 5.59. Finally, we have the LR model, with MAPE of 5.66.

Figure 7 shows us how our forecast accuracy has changed over time. This graph, like Figure 5, is another graph that looks a little odd in recent years due to the database format change. Taking that into consideration, we can see that up until about 2009, the ANN and LR models have drastically improved, and the accuracy of the DPP model has stayed somewhat the same over time.

Figure 8 shows us the accuracy of different forecast numbers. If an LDC chooses to do so, they can run GasDay's forecasts up to six times a day. The forecast number coincides with the number of times the forecaster ran during that day. We can see that as the day goes on, the forecasts generally become more accurate.

The final program that I ran over the data shows the forecast accuracy of the different

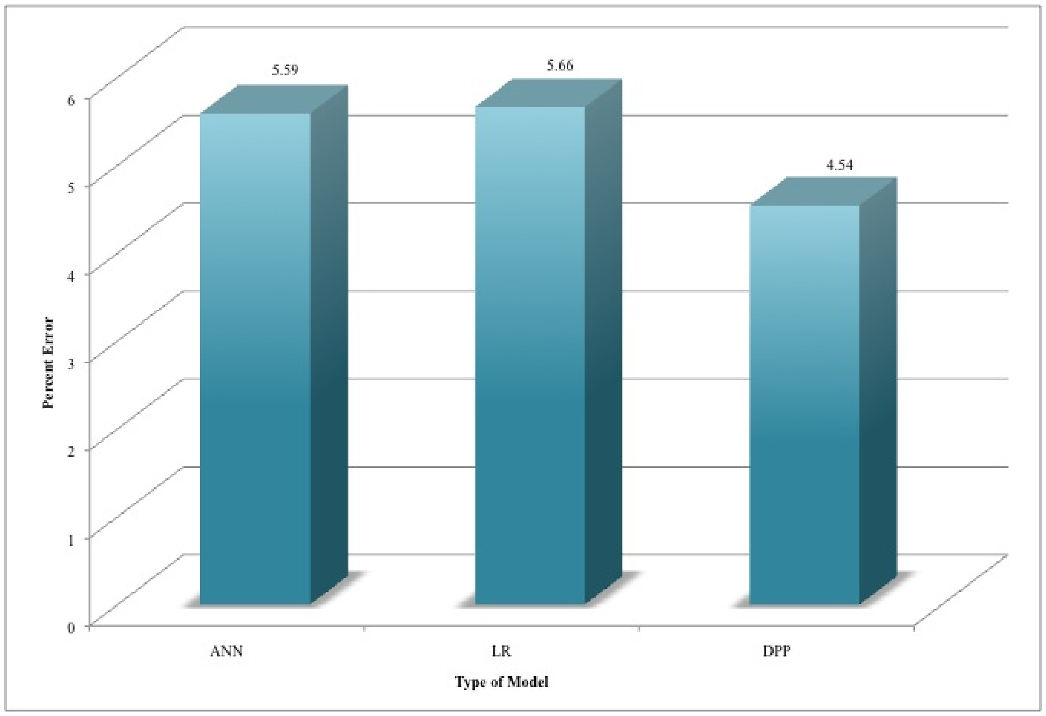


Figure 6: Most accurate type of model

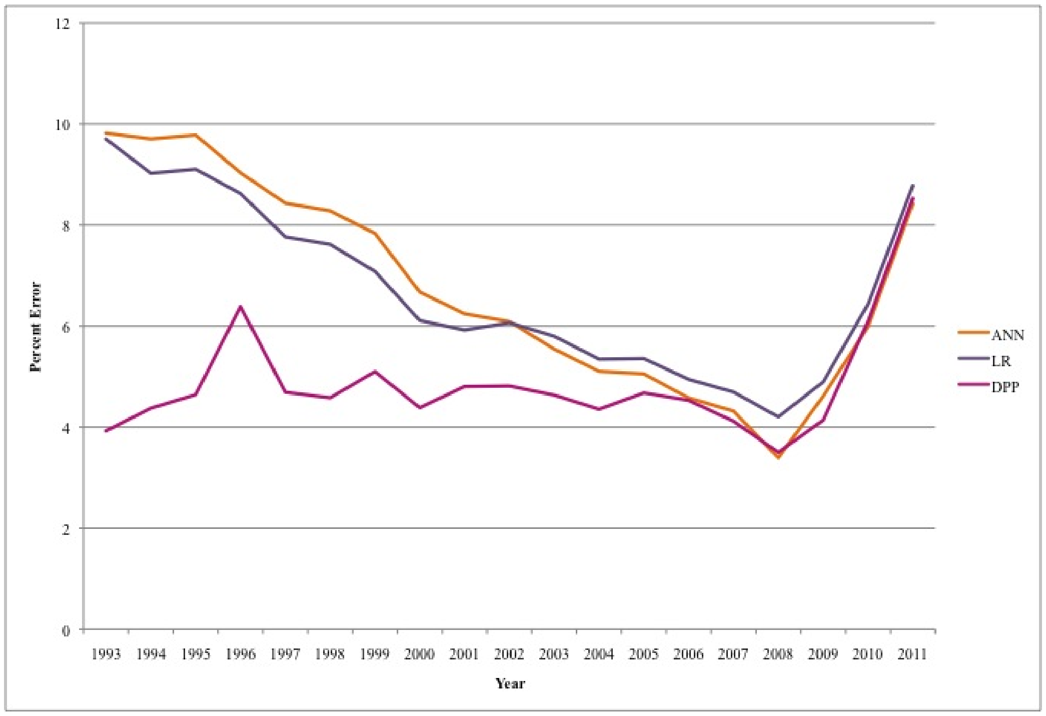


Figure 7: Forecast accuracy over time

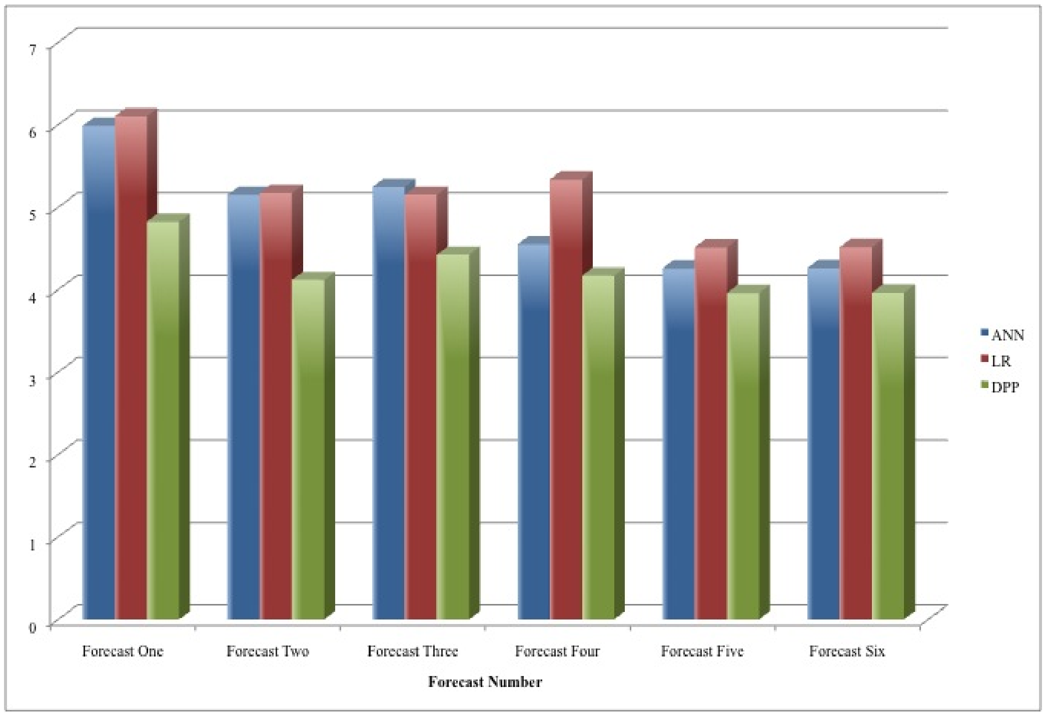


Figure 8: Most accurate forecast number

forecast numbers for each LDC. Overall, we can see some very interesting trends from these graphs, and it makes GasDay curious to find out more about all of our data. Using the MapReduce and Hadoop frameworks seems to be an ideal way to process and analyze a large amount of data, which is why you are reading this paper. This project will continue on to grow and improve on what Tom, Samson, and I have done so far. Finally, I will tell you where you can find the information you will need.

6 Wiki and Other Resources

The Wiki I created during my Summer with the GasDay lab is intended to help you along the way. To access the Wiki, you must be connected to GasDay's network in the lab. The main page is located at www.wiki.gasday.com. When you reach the main page, you will click on the section at the top labeled [Student Projects and Research.] To access any of these sections, you must have login credentials that are approved by GasDay. From there you will click on the link located in the section labeled “Research Projects” that is labeled [Using MapReduce and Hadoop.]

The Wiki contains almost everything that you will need to either start this project over or continue on where I left off. Some examples of material listed on the Wiki are the tutorials I used to set up the Hadoop cluster, my system administrator log that goes along with the tutorials, great Python tutorials, helpful Ubuntu command line tricks, important websites, and all of the sources I used to understand the MapReduce and Hadoop frameworks.

Along with the Wiki, I copied everything related to my project onto a CD for Dr. Corliss. Everything is also copied to the drive on the GasDay network that is specified in the Wiki.

*References

[1] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, January 2008.

[2] J. Dean, S. Ghemawat, and G. Inc, “Mapreduce: simplified data processing on large clusters,” in *In OSDI04: Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation.* USENIX Association, 2004.

[3] D. Kusnetzky. (2010, February) What is “big data?”. Access date: June 27, 2012. [Online]. Available: http://www.zdnet.com/blog/virtualization/what-is-big-data/1708

[4] M. Nielsen. (2009, January) Write your first mapreduce program in 20 minutes. Access date: June 5, 2012. [Online]. Available: http://michaelnielsen.org/blog/write-your-first-mapreduce-program-in-20-minutes/

[5] M. Noll. (2007, August) Running hadoop on ubuntu linux (single-node cluster). Access date: June 1, 2012. [Online]. Available: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/

[6] R. C. Taylor, “An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics,” *BMC Bioinformatics*, vol. 11, pp. 1–6, July 2010.