Large-Scale Exploratory Analysis, Cleaning, and Modeling for Event Detection in Real-World Power Systems Data

Ryan Hafen
Pacific Northwest National Laboratory
ryan.hafen@pnnl.gov

Tara D. Gibson
Pacific Northwest National Laboratory
tara@pnnl.gov

Kerstin Kleese van Dam
Pacific Northwest National Laboratory
kerstin.kleesevandam@pnnl.gov

Terence Critchlow
Pacific Northwest National Laboratory
terence.critchlow@pnnl.gov

ABSTRACT

In this paper, we present an approach to large-scale data analysis, Divide and Recombine (D&R), and describe a hardware and software implementation that supports this approach. We then illustrate the use of D&R on large-scale power systems sensor data to perform initial exploration, discover multiple data integrity issues, build and validate algorithms to filter bad data, and construct statistical event detection algorithms. This paper also reports on experiences using a non-traditional Hadoop distributed computing setup on top of a HPC computing cluster.

Categories and Subject Descriptors

J.2 [**Computer Applications in Physical Sciences and Engineering**]: Engineering

General Terms

Data Analysis

Keywords

Power Systems, Phasor Measurement Unit, Divide and Recombine, Hadoop, R

1. INTRODUCTION

In application areas involving large-scale distributed sensor networks, prior to deploying algorithms over high performance computing and networking infrastructures, it is important to consider the validity of the algorithms and of the sensor data itself. Real-world sensor data presents many challenges such that algorithms developed from the study of simulations, physical models, or experience of domain experts are generally inadequate when applied directly to the data. Additional work is frequently required to validate or adapt these approaches to the reality presented by the captured data, or to discover better models and algorithms.

Building and validating algorithms that capture phenomena of interest in real-world large-scale data sets is not a simple task. As we don't know what to expect in the data, analysis is typically exploratory and iterative and requires a great deal of flexibility in numerical and visual methods applied. Also, as some phenomena are rare, it is difficult to look only at small subsets of the data, and we need tools that allow us to apply analytical methods at scale with great ease.

Exploratory analysis and data cleaning is an incredibly important but difficult and time consuming task, which in the experience of practitioners, constitutes 80% of the effort that determines 80% of the value of the ultimate analysis results [5]. For large data, the task is even more difficult because the traditional tools do not scale. Regardless of size, to be effective we must be able to look at all the data in great detail.

In this paper, we illustrate the use of a methodological analysis approach and computing environment to to explore and clean a 2 TB power grid data set. This data comes from a distributed network of Phasor Measurement Unit (PMU) sensors throughout the grid reporting measurements at high temporal resolution. The data is being used to support real-world decision support systems for wide-area power grid situation awareness, reliability, and emergency management. There will be many down-stream analyses applied to this data and hence its integrity and the integrity of the algorithms is critical.

PMU data is a relatively new data stream for power engineers, and as a result the characteristics of the data stream are not well understood. In order to effectively explore, clean, and analyze the data, we needed an analysis approach that would allow us to identify uncommon patterns within the data and validate these patterns with domain experts, but which did not require the experts to be able to model these events in advance. Exploratory data analysis met this requirement.

To accomplish our analysis, we use an approach called Divide and Recombine (D&R), using a software and hardware environment that allows us to write code in the R statistical programming environment with distributed computations being executed on a Hadoop cluster. This provides the flexible rapid development and iterative analysis capabilities required for our analysis as well as the required scalability. As this is an important aspect of the analysis, we discuss this D&R and its computing environment in Section 2. In section 3, we describe the data and Section 4 outlines our analysis results. Finally, we presents results of a scalability study of our environment applied to replications of this data in Section 5.

2. COMPUTING ENVIRONMENT

Proper tools are required to perform comprehensive data analysis at scale. An interactive environment is required, and to achieve this, an analyst needs to be able to 1) flexibly and rapidly develop and update algorithms, methods, and visualizations, and 2) apply these algorithms against the entire data set and receive results in a reasonable amount of time. Hardware and software environments must facilitate these requirements. In this section we introduce the Divide and Recombine (D&R) approach to large-scale data analysis, describe its components, and provide information on the implementations we have successfully used in our analyses.

2.1 Divide and Recombine

Divide and Recombine (D&R) is a statistical approach to analysis of large complex data [7]. In D&R, The data are divided into subsets in a statistically meaningful way, analytic methods are applied to each subset in an embarrassingly parallel manner, and the results of each method are recombined to form a result for the entire data. Analytic methods can be either visual or statistical. For a visualization method, the recombination is a visual display comprised of panel views of each subset. For statistical methods, the recombination result is numerical. A simple example of a statistical recombination method is applying a logistic linear regression independently to each subset and combining the coefficients from each model fit by taking their mean. There is much ongoing research in valid recombination approaches. An example of an approach that fits into the D&R analytical recombination paradigm and covers a wide range of statistical methods is the Bag of Little Bootstraps [11].

Data can be divided in different ways for different analysis tasks. Many divisions are determined by splitting the data according to combinations of categorical variables – a natural division choice. Examples of such divisions are partitioning by location, time, or experimental subject. Another division approach is a random or stratified random partitioning of the data. Division operations are typically very expensive, as the entire data set is being read in and written back out into a new partitioning. Once a division is created, it persists and is hit repeatedly with recombination methods throughout the course of the analysis. Recombinations are typically much less expensive, since they typically result in a great reduction of the data. The general approach in D&R is to choose a good division and then apply multiple recombinations to that division.

The ultimate goal of D&R is to provide a statistically sound methodological approach to large complex data that provides computational feasibility and practicality, making available libraries of 1000s of single-core statistical method implementations that might be infeasible or impossible to rewrite in a parallel version. This point is critical particularly in open-ended data analyses, where the correct models and methods to apply are determined by the data, and time cannot be wasted implementing a sophisticated exact parallel version of an algorithm only to find out that it is not adequate for the data. The tradeoff for increased productivity and greater access to libraries of statistical methods can sometimes come through an approximate analytical result rather than an exact one, although in our various experiments we have repeatedly found this discrepancy to be extremely negligible.

2.2 D&R Storage and Computation

The foundation of D&R is key-value data stores for distributed storage of the divided data, and MapReduce for parallel computation on the data. Divided data are stored as key-value pairs, where the value of each pair is one division of the data. This provides flexibility in the structure of the data objects for each subset, and is a natural storage mechanism for MapReduce processing.

MapReduce provides a versatile high-level parallelization to solve many data-intensive problems through use of user-specified Map and Reduce functions [6]. A MapReduce job begins by taking the input data, which is a collection of key-value pairs, and applying a Map function to these pairs independently. Each input pair is assigned to a processor, which computes the map function and returns a new key-value pair. All output values associated with a specific key are grouped together and processed by a Reduce function, which produces a collection of final key-value pairs. Reduce functions are also independently computable and are executed in parallel. This process is illustrated in Figure 1.

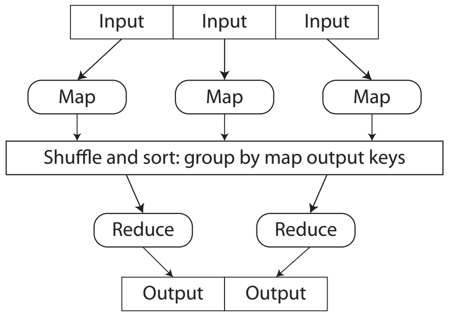

Figure 1: Overview of the MapReduce paradigm.

MapReduce is a natural processing engine for D&R. Partitioning of the data in the division step is achieved through one or more MapReduce operations. Recombination is usually a more simple MapReduce operation where sometimes the reduce phase is omitted or is a simple collation of map outputs. The majority of analyses in D&R are approached in terms of thinking about divisions and recombinations as opposed to MapReduce, and a domain-specific language that hides the MapReduce details is under active development that simplifies the coding of D&R tasks significantly [9]. However, sometimes we are interested in division-agnostic all-data summaries or computations, in which case we use MapReduce directly.

2.3 D&R Software Environment

In this work, we use the Hadoop implementation of MapReduce supported by the Apache Software Foundation [8]. It is the standard open-source MapReduce framework. It hides complexities of job scheduling and tracking, data distribution, system architecture, heterogeneity, and fault-tolerance. Hadoop also provides a distributed key-value storage system, the Hadoop Distributed File System (HDFS).

While Hadoop is a good choice for distributed D&R computations, writing analysis code in its native language, Java, does not satisfy the requirement of rapid prototyping and interactive programming necessitated by the iterative analysis process. The interactive interpreter-based R statistical language provides an excellent environment for rapid prototyping of data analysis routines. R has excellent capabilities for exploratory analysis, visualization, and modeling, including a wealth of statistical routines and over 4000 user contributed packages [4]. The R and Hadoop Integrated Programming Environment (RHIPE) is an open-source effort that allows data analysts to write MapReduce code in R to be processed by Hadoop [15]. This package manages all the details of Hadoop MapReduce, including data input/output, data serialization, job submission, and job monitoring, without ever leaving the R console. While R and Hadoop are not new technologies, their combination is a breakthrough for promoting detailed, comprehensive analysis for data analysis in massive databases, and ultimately lets the user focus on thinking about the data rather than the complexities of the distributed computing environment.

2.4 D&R Hardware Environment

Traditional Hadoop clusters allow for both on-demand compute resources and persistent, always-accessible storage. Storage is facilitated by the HDFS, and on-demand computing is facilitated by Hadoop's inherent multi-user architecture through job and task scheduling.

Unfortunately, we were not able to access a dedicated Hadoop cluster for our work. Instead, we utilize an institutional HPC cluster with modifications that get us closer to the requirements of interactive data analysis. This cluster consists of over 600 32-core nodes, connected to a 4 PB Lustre data store, although a typical allocation used for this work was a substantially smaller subset of the nodes. Our data resides on the Lustre high performance, distributed file system, which is accessible by all nodes, as opposed to being distributed across local disks as is typical for HDFS. High throughput network connections enable remote access at speeds theoretically comparable to local disk.

Launching a job consists of running a script that allocates the nodes, and instantiates a Hadoop cluster across the nodes. This process requires only a matter of seconds and therefore does not add much tedium to an interactive analysis requirement. Once the Hadoop cluster-on-demand is running, we login to the master node, launch R, load the RHIPE package, and have an interactive environment to run our MapReduce job. Although nodes are allocated and deallocated for each job, the data stored on Lustre persists.

Throughout the course of our analysis, we ran hundreds of RHIPE MapReduce jobs over the 2 TB data set using anywhere from 10 to 50 compute nodes, configured to execute no more than 20 simultaneous tasks per node. Run times ranged from 10–60 minutes. Most operations involved a read of the complete data set, with the associated computation time being negligible in comparison to the I/O. The longer-running jobs involved large amounts of data being shuffled between the map and reduce phase, as well as large outputs written to disk. For example, the division of the data into true 5-minute blocks. Beyond the basic Hadoop tuning parameters, we have not made extensive investigations into tuning the performance of our jobs.

3. POWER GRID DATA

The analysis presented here is part of a larger project in which we are building a queryable meta-data repository of interesting events in large-scale power grid data for use in power grid decision support systems. The data analysis aspect of the project contributes to the meta-data repository by finding these interesting events.

This section provides a description of the power grid data we working with as well as the pre-processing steps and initial division we created for analysis.

3.1 Data Description

The data in our analysis consists of measurements taken by Phasor Measurement Units (PMUs) residing at various substation locations on the power grid. Each PMU contains between 3 and 11 sensors, with each sensor measuring a variable such as frequency, voltage, current, or phase angle. These sensor variables, as well variables such as status flags and location-specific meta-data are disseminated in a continuous data stream, reporting measurements at 30 times-per-second. PMUs use a highly accurate global clock to ensure all data records are time-synchronized.

Our data set spans $\sim$1.5 years for 38 PMUs. At 30 measurements per second, this amounts to about 1.5 billion time points at which measurements are taken. The number of individual measurements taken by a single PMU at a single time point varies from 6 to 22 (some variables are tuples). Thus a single PMU can report up to 33 billion records over this time period. In binary format, the data is 1.9 TB in size. While this is a relatively small data set by today's standards, PMU data is expected to grow significantly in the coming years, with data sets reaching the 100TB – 1PB level. Thus, the scalability of analysis algorithms is a significant concern.

We focus on frequency throughout this paper. Each PMU reports one frequency series, giving a total of $\sim$53.7B sensor records to work with. Frequency is a measure of the cycles per second of current flowing through the wire. It is an indicator of the grid's ability to respond to changes in supply and demand of electricity. Frequency should always be close to 60 Hz.

3.2 Data Preparation

The raw data consists of about 157,000 binary files, each typically containing all PMU measurements for a 5-minute time interval. This raw data is transformed into a Hadoop-friendly key-value pair format, with the key being the start time of the reading and the value value being a matrix of the data corresponding to that file: each row corresponds to a specific time and each column corresponds to the value of a specific sensor's measurement for one variable at that time. The resulting matrix typically has 9000 rows, corresponding to the number of 1/30th second records generated every five minutes, and 555 variables, corresponding to the 6–22 measurements recorded by each of the 38 sensors that we are tracking.

The events we are interested in finding in this data set are localized in time, and their lifespan is typically on the order of seconds. As such, time is the obvious choice for the conditioning factor in our division, allowing us to create data subsets consisting of temporally contiguous data. We determined a reasonable data size for an individual subset of data to be 5 minutes. This range is long enough to capture events and short enough that the individual block size is not too large in memory. The raw was already almost divided in this fashion, but the data did not always partition along 5-minute intervals, so we ran a division MapReduce job to push each observation into the appropriate 5-minute time block. For example, an observation that occurs at 2010-01-01 12:32:23.5 would be assigned to the group starting with time 2010-01-01 12:30:00.0. The 5-minute time as a numeric Unix time value was chosen to be the output key, with the associated data as the value. The output of this MapReduce job is a special type of Hadoop file type called a map file: map files can be queried by key, so by setting the key to the date and time allows the data of interest to be easily and quickly retrieved.

To keep the data compact, we stored frequency as an offset in thousandths from 60 Hz, e.g. a value stored as -1 corresponds to a frequency of 59.999 Hz. This is how the raw data is stored, and this is the finest resolution at which the frequency was reported. Storing the data in this way allows us to use an integer instead of a floating-point number, which reduces the file size. The cost of making the conversion to Hz on each read is more than offset by faster I/O for a smaller file.

In this paper, each of the 38 PMUs is labeled using a two-letter symbol.

4. ANALYSIS METHODS AND RESULTS

A general approach to exploring a new data set in D&R is to first compute and visualize summaries of the subsets with a simple summary recombination. We study the properties of these calculations and identify subsets of the data that demonstrate interesting characteristics and explore those data subsets in detail. In most cases, these steps generate data sets small enough that we are able to perform the detailed analysis on a local workstation. If some subsets of the data represent an event of interest, such as bad data, we pull some of the subsets to our local R session, apply methods to try to capture the behavior of interest, then apply that method across the entire data set, using RHIPE, and evaluate the results. We then repeat this process, generating new statistics based on our previous analysis of the data.

Initial inspection of the PMU data gave a strong indication that bad data beyond what we were initially led to believe was present in the data. This prompted an in-depth investigation for bad data, and led to identification of several data quality issues. This work had to be carried out prior to any subsequent event detection algorithm development, as the bad data severely interferes with the event detection algorithms and can lead to a large number of false-positives. In our data cleaning efforts, we took an extremely careful and conservative approach to filtering data, ensuring that there is effectively no chance that we are eliminating valid data records when performing our data cleaning.

This section describes three classes of data cleaning techniques we discovered and applied against our data set, and then describes two event detection algorithms we developed after cleaning the data. The purpose of this section is to illustrate the necessity and nuances of analyzing real-world large-scale datasets as will as illustrate the D&R toolset in action.

4.1 Overlooked Flags

Each PMU at each time point reports a flag indicating the status of the measurement at that time. Domain experts stated flag 130 indicates a bad data point being recorded. When a bad data flag is present, the corresponding frequency value is reported as -1 (thousandth offset from 60 Hz), which unfortunately translates to a very legitimate frequency value, 59.999 Hz.

One of our initial exploratory computations was to look at the distribution of frequency for each PMU. A good way to do this is to compute quantiles and look at quantile plots. Taking advantage of the frequency being reported as a discrete integer value (thousandth offset from 60 Hz), we applied a division-agnostic MapReduce to compute quantiles by tabulating the number of observations at each unique frequency offset for each PMU. The Map task tabulates the frequency values for each PMU and emits the PMU and frequency value as the key and the count as the value. The Reduce task collects the counts for each unique key (PMU) and sums them. This results in a histogram of frequency, which we can essentially integrate to obtain quantiles. For many PMUs, the distribution of frequency looked like a Gaussian distribution, but some exhibited a behavior in which value of 59.999 Hz occurred much more frequently. Figure 2 shows a normal quantile plot of frequency for two PMUs. The quantiles for the data are plotted against the quantiles for a normal distribution with mean and variance matching that of the data. PMU AR represents a typical PMU, while PMU BA illustrates data with the more abundant 59.999 Hz value, making up over 40% of the observations.

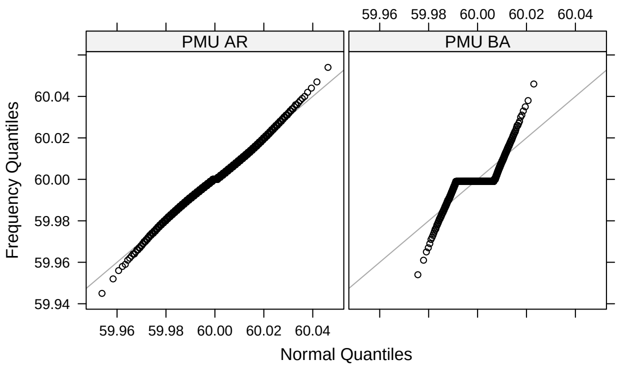

Figure 2: Normal quantile plot of frequency for two PMUs.

Since flags for bad data report a value of 59.999 Hz, we suspected that taking flags into account might provide insight. We performed the same tabulation of frequency counts, but this time for each PMU and flag combination. Figure 3 shows the distribution of frequency deviation by flag for PMU AZ. The dashed line is used to highlight the distribution corresponding to a frequency offset of -1. For PMU AZ, other than flag 130 we see three other flags for which effectively all observations are -1, suggesting that these flags also indicate bad data. We validated that these flags are indeed bad data indicators by viewing the frequency time series where these flags are set and seeing that the corresponding -1 values deviate significantly from neighboring time points. We also obtained confirmation from domain experts.

It is interesting in Figure 3 that the frequency distribution for flags 2 and 3 looks Gaussian except for the spikes at zero. In both cases, there are no values for a frequency of -1.

Accounting for the extra flags removed about 8.135B billion points from the data. This figure is skewed by one PMU which had 100% bad data. Not counting that PMU, we removed 6.728B records.

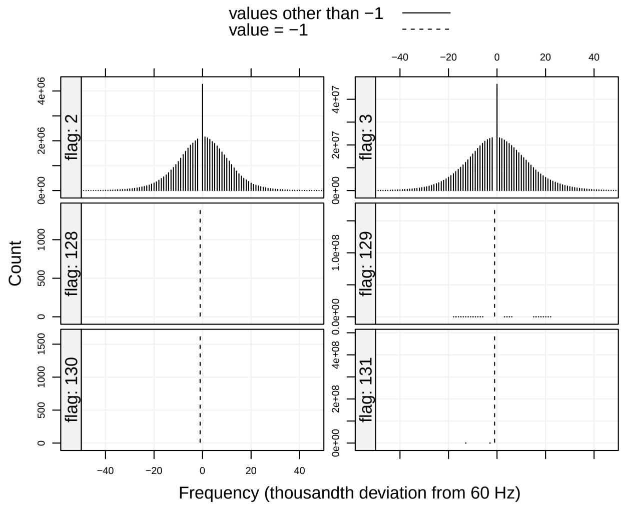

Figure 3: Distribution of frequency offset counts for PMU AZ, broken down by flag.

4.2 Repeating Values

While we dealt with abnormally high levels of -1 values due to bad flags, we also noticed some cases of an unusually high number of zero values, both in quantile plots and frequency distribution plots. Close-up looks at frequency time series for individual subsets revealed what appears to be abnormally long sequences of repeated zeros. The frequency signal should be changing in response to constant changes in the environment and should not remain constant for very long. Thus, long sequences of repeated values are suspect.

To better understand the normal sequence length of repeated zeros in the data, we applied a recombination method that calculated the distribution of the run length of repeated zeros for each PMU within each 5-minute block. Because each computation occurs on a single 5-minute window, the largest possible sequence length is 9000 (5 min * 30 records / second). For our purposes, this is acceptable because we are looking for impossibly long sequences of values, and a value repeating for even one minute or more is impossibly long. We do not need to accurately count the length of every run, only identify those repetitions that are too long to occur in a correctly working system. Further, if a sequence happens to slip by the filter, for example because it is equally split across files, there is little harm done. An exact calculation could be made using MapReduce, but the additional algorithmic complexity is not worthwhile in this case.

Figure 4 shows the zero run length distribution for two PMUs. This distribution is very long-tailed and is thus displayed on a log-log scale. The distribution for both PMUs is very similar, with the count trailing to zero near the run length of 10 seconds. However, we see that PMU AS has observations occurring outside of the region where the distribution trails to zero.

We noticed that beyond the very short sequences of repeats, the tail of the run length distribution behaves in a way similar to the tail of a geometric random variable with mass function

$$P(x = k) = \frac{p(1-p)^{k-1}}{(1-p)^{N-1}}, k = N, N+1, \ldots \tag{1}$$

The solid black line in Figure 4 indicates the fit of the geometric tail distribution ($\hat{p} = 0.04$ and 0.03 for AS and BI respectively). We can use this observation to set limits based on the estimated geometric parameters for which we would likely never expect to see a run length exceed. PMU AS in Figure 4 shows several cases well beyond the tail of a legitimate distribution. These sequences of well over 45 seconds correspond to bad data records.

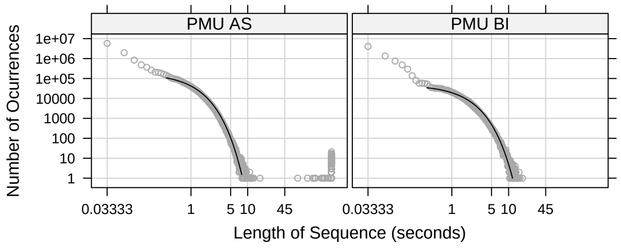

Figure 4: Distribution of run length of repeated zeros for two PMUs.

This is a good example of how data analysis can provide information that is perplexing to subject matter experts. A consecutive run length of even 3 seconds for a frequency measurement was regarded as inconceivable by experts, but based on our analysis it is plausible and does occur.

We repeated this analysis for all repeating value sequences (not just zeros), and eliminated 123 million records from the data set.

4.3 White Noise

Applying the event detection algorithm described in Section 4.4 led to the discovery of more bad data. A prevalent example was where one PMU reports what appears to be white noise while the other PMUs are reporting valid data. What made these hard to detect in prior exploratory analysis was the fact that the white noise typically fell within the acceptable frequency range. Figure 5 shows a 10-minute time series plot of frequency for three PMUs. PMU AX exhibits white noise. The other frequency series tend to behave as highly autocorrelated processes fluctuating around 60 Hz.

We verified with domain experts that the white noise indeed is indicative of bad data. To remove white noise data, we employed the Box-Ljung test statistic [12]. This test determines whether any of a group of autocorrelations is significantly different from zero. With white noise, all autocorrelations should be close to zero. For a time series of length n with $\hat{r}_k$ denoting the sample autocorrelation at lag k, the test statistic is computed over m lags as

$$\tilde{Q}(\hat{r}) = n(n+2)\sum_{k=1}^{m}\frac{\hat{r}_k^2}{n-k} \tag{2}$$

and is approximated by a χ^2 distribution. To get a good

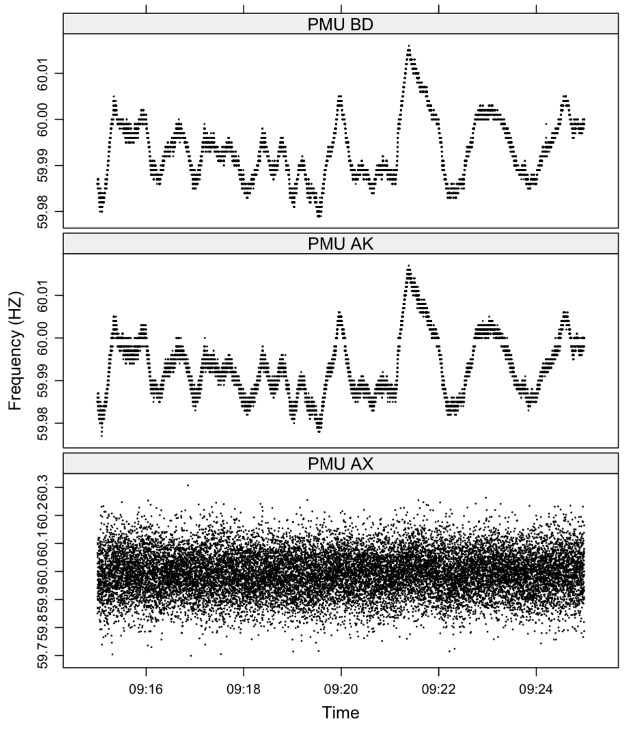

Figure 5: 10 minutes of frequency data for three PMUs.

sampling of lags, we chose $m = 15$ and applied the test statistic to each frequency series within each 5-minute block. Tests that report significant p-values correspond to the legitimate, highly-autocorrelated frequency series, and in our case were virtually zero, while non-significant (using $p > 0.05$) correspond to white noise. Using this rule, even after all of the other bad data had been filtered, we removed 25 million records.

4.4 Frequency Deviation Event Detection

After performing the initial data cleaning, we began looking for specific events of interest. From discussions with domain experts, we learned that one type of interesting behavior occurs when the difference between time-synchronized frequency for two locations is significantly high. As seen in Figure 5, the frequency at different locations typically behaves in the same way at any given time (see PMUs BD and AK). In order to determine what signifies a significant frequency deviation between two frequencies, we investigated the distribution of the difference between all pairwise combinations of PMUs. We calculated several statistics on the differences, including quantiles near the tail end of the difference distribution. Instead of a simple tabulation of data within each subset, we are transforming the input data into pairwise groupings and then performing the calculations across each group. This is a good example of something that is trivial to implement in MapReduce (and all the more simple using RHIPE), but which might not be so straightforward otherwise.

There are 37 sensors reporting good frequency data, so there are $\binom{37}{2} = 666$ possible PMU pairs. Figure 6 shows the sorted 99th percentile value for the distribution of the frequency difference for each pair. It is interesting that certain pairs have a very tight distribution at the lowest end, where some time-synchronized pairs are within 1/1000th of each other 99% of the time, while other pairs have a frequency difference whose tail extends five times further. This is due to geographical distance between the PMUs. A multidimensional scaling using the 99th percentile as a distance metric constructed a map that matched the geographical map of PMU locations quite closely.

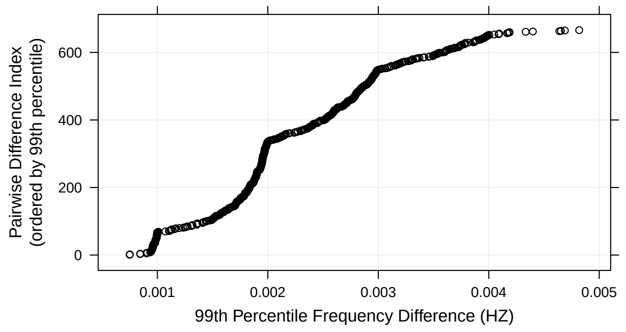

Figure 6: 99th percentile of absolute pairwise frequency difference for 666 PMU pairs.

We used the median and standard deviation of the 99th percentile calculated at each 5-minute interval of the pairwise differences to choose limits beyond which frequency differences are deemed to be significant. We verified these cutoff limits visually. The initial event detection algorithm was to find every case when any pairwise difference at any time point exceeds the specified limit for that pair. This resulted in the detection of spurious singleton differences. However, the cases where the deviation is persistent across time are significantly more interesting. Therefore, we adapted the algorithm to filter the results based on the length of time for which the difference is persistently of the same sign, requiring a duration of at least 3 seconds. A recombination job took each PMU pair inside each 5 minute interval and looked at the frequency difference for sequences matching these criteria, recording details when events were found.

The algorithm returned 73 events, which can be grouped into 6 categories. A representative event of each type is shown in Figure 7. For time scales in the sub-minute range, the x-axis is labeled in units of seconds (e.g. 00, 15, etc.) whereas when the event spans multiple minutes, the x-axis is expressed as time (e.g. 16:45). The y-axis labels (frequency) are omitted in the interest of space but are comparable for each event. Each line represents a different PMU.

Event 1 is an example of a generator trip. When a generator trip occurs, the effect is a sudden drop in the frequency across the grid, which gradually returns to a stable state after automated controls kick in. These events can cause opposing oscillations across groups of PMUs as seen immediately after the initial frequency drop.

The general pattern in Event 2 is characterized by one PMU jumping off from the pack and following the general shape of the others with a positive or negative offset. Typically a frequency disturbance at one location impacts the frequencies at all other locations, which may indicate that this type of event is a more sophisticated "bad data" case. However, it is unique enough to warrant extra scrutiny. Sub-

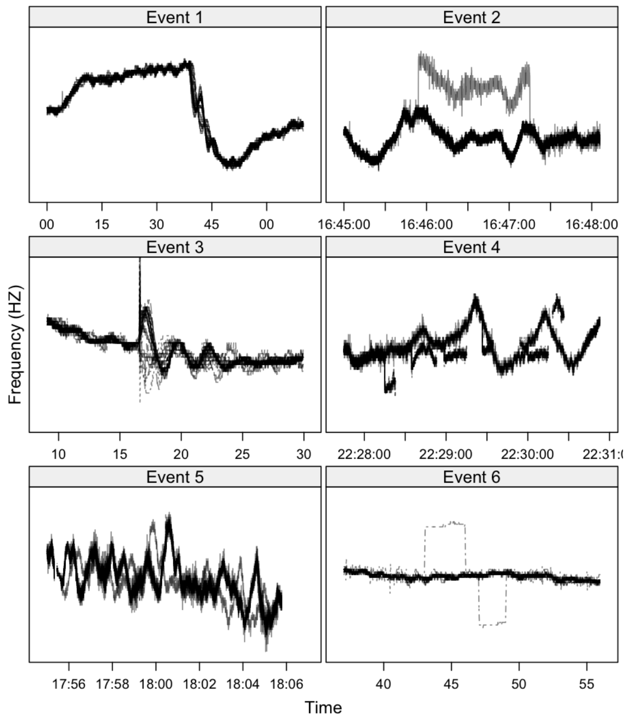

Figure 7: Six examples of interesting events from the frequency deviation event detection algorithm.

ject matter experts are currently working on understanding this type of event. Similar to Event 2, the pattern in Event 6 is characterized by a jump in one direction directly followed by a jump in the opposite direction.

Event 3 is characterized by a spike in activity (the spike well exceeds the plot region) followed by opposing oscillatory behavior for different groups of PMUs. What makes this type different from Event 1 is that it is not set off by a sudden drop in frequency, but by a spike. According to domain experts this is indicative of a line fault.

Events 4 and 5 are unique instances across the entire data set. Event 4 shows groups of PMUs jumping off from the main frequency signal and sporadically missing data in chunks of time. Event 5 shows a single PMU operating at a different frequency that is time-shifted from the rest for about 10 minutes, a likely case of a bad timestamp. The behavior for both of these events could potentially be indicative of a grid island, where some groups of locations are operating independently of the others. However, based on previous island events that subject matter experts are familiar with, these events do not seem to have the same characteristics, and have been deemed to be another special case of bad data.

Overall, the results show events where the interesting patterns such as sudden drops or opposite oscillations occur at the sub-second level. This underscores the importance of looking at the entire data set and not resorting to a more coarse time aggregation, such as the mean frequency over multiple seconds or minutes, in which case many of these features would be washed out. This is only possible with a scalable environment capable of analyzing the entire data set.

It worth noting that prior to applying the data cleaning filters, the application of the frequency deviation event detector to the raw data returned tens of thousands of events. After account for bad data, there are only 73 distinct events – a dramatic difference that underscores the importance of understanding the data and accounting for data integrity.

4.5 Generator Trip Event Detection

As noted, the frequency deviation detection algorithm turned out several events that were identified as experts to be generator trips, such as Event 1 in Figure 7. We studied the properties of these confirmed generator trip instances to determine features and algorithms to detect trips.

As we saw in Event 1 of Figure 7, generator trips are characterized by the sudden and steep decline in frequency that occurs when a power generator goes offline, followed by a gradual increase back to a stable frequency. We captured this behavior by segmenting the data into increasing and decreasing sequences and computing features about these sequences, such as the steepest slope in each segment. Determining the low-frequency change-points of slope in the frequency series was done using the robust nonparametric regression method loess [3], which is essentially a low-pass filter that smooths out high-frequency fluctuations.

Nonparametric local smoothing methods are not reliable at endpoints, and therefore applying the method independently to each 5-minute subset would yield poor smoothing at the beginning and end of each 5 minute segment. Therefore, we developed a more suitable division, in which we added an extra 30 seconds to the beginning and end of each 5 minute period, resulting in overlapping divisions. We chose 30 seconds because a 14 second moving time window for the local smoothing was determined to be an appropriate choice, and having an extra 30 seconds on each side would provide enough extra data to get good results with some leeway in case we later settled on a slightly bigger window size.

Operating on this division, we applied the smoothing and sliced the data into monotonically increasing or decreasing segments. An illustration of the smoothing and segmenting is shown in Figure 8. The gray points in the background represent the actual frequency values for the 38 PMUs. The black lines represent the smoothing, with the segment breakpoints indicated with a vertical line separating increasing from decreasing, and with decreasing segments drawn with a thicker line.

After obtaining segments, we collected features for each segment with a recombination procedure that calculated features such as the maximum slope of the segment, the change in frequency from beginning to end of the segment, and the duration of the segment. We then visually explored the distributions and relationships between these features in many ways, mostly through the use of hexagonally binned scatterplots [2]. For the maximum slope feature, we noticed a somewhat symmetric distribution with an additional small mode on both tails. We selected subsets of our data that corresponded to segments with slopes in the lower-tail mode of the distribution and gave these more intense scrutiny, since we expect generator trips to have a large negative slope. We selected about 2000 time windows containing very negative slopes and presented them to domain experts to categorize as generator trips or not. This oracle-classified data along with randomly-selected non-generator trip time windows became a training set against which we applied machine learning algorithms to classify events as generator trip or not.

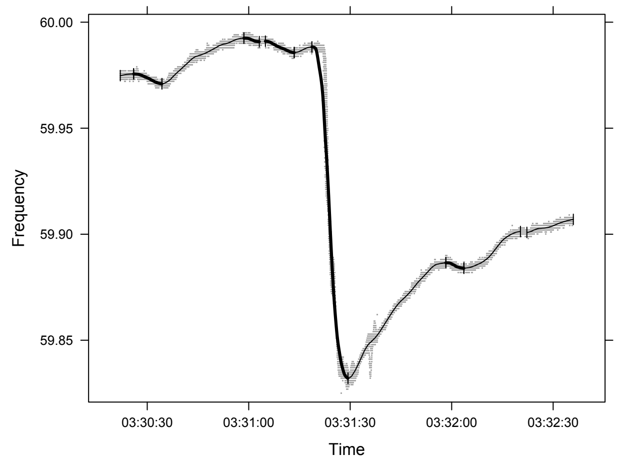

Figure 8: Illustration of smoothing and segmenting frequency data for generator trip feature construction.

After studying several classification approaches, we found that a simple rules-based cutoff method based mainly on the maximum slope feature yielded excellent results while being very easy to interpret and implement in real-time processing system.

This event detection algorithm was implemented as a proof-of-concept in an agent execution platform, Volttron [1]. Once an algorithm has been validated and is ready to be deployed, a different set of considerations come into play from a HPC point of view, and the ability to rapidly process large volumes of historical data becomes less of a concern. But had we not been able to work with ease with the large amount of historical data, we would not have been able to develop and validate the algorithm.

5. SCALABILITY STUDY

Although the size of the actual data we analyzed in this work is only 2 TB, we anticipate a significant increase in the data volumes of the future as many more PMUs are rolled out. To evaluate how our infrastructure will work in that new environment, we replicated our original data set to create test data sets of 4, 8, 16, and 128 TB. 128TB represents over 43,000 PMU months of data, and approximates the anticipated monthly data stream from all PMUs within the US when the grid update is completed in the 15 years. Hadoop has been routinely used for data sizes much larger than this, but our goal is to (a) ensure that nothing in the design of RHIPE inhibits scalability, and (b) understand how our system configuration, described in Section 2.4, scales.

We vary the number of nodes used in computation. We tested node configurations of 8, 16, 32, and 64 nodes, and in the case of the 128 TB dataset, 128 nodes. For each of these, an additional node to function as the Hadoop namenode was added. Although the nodes have 32-core processors, we ran the tests at an underutilized rate of 10 maximum map tasks and 10 maximum reduce tasks per node. There are a large number of Hadoop parameters and we note that parameter settings we chose are not necessarily optimal.

We ran a simple analysis on the data, which tabulates the flag frequencies and aggregates the results for each PMU. The map step in this task greatly reduces the data, so much less data is being written to disk than is being read. This is typical for most of our jobs once the data was divided properly.

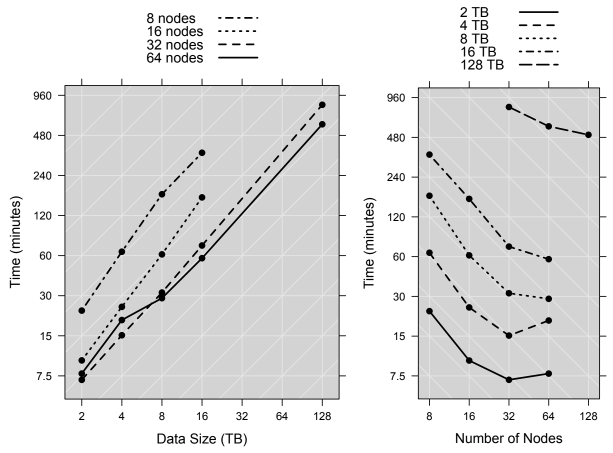

Figure 9: Scalability test results: Time vs. Data Size, by Number of Nodes (left); Time vs. Number of Nodes, by Data Size (right).

Two graphs showing the timings for running this job are shown in Figure 9. The plot on the left shows that for a given set of nodes, the compute time increases almost linearly with an increase in the data size. Looking at the data differently, however, shows that our infrastructure is not providing the ideal scaling we desire. The plot on the right shows for a fixed data set size, the reduction in computation time is not commensurate with the additional resources provided. For the smaller data sets, this result is entirely expected because the job quickly becomes I/O bound and the analysis time is dominated by the combination of the fixed overhead cost of starting additional compute nodes and the constant time required to stream the data off of the file system. The reason for the sub-linear scaling is less obvious for the larger data sets. We suspect that this is largely due to the impact of using the Lustre file system to store the data as opposed to local disk. This increases network bandwidth and limits the throughput an individual node can obtain. While other factors are also likely impact to these timings, for example the jitter caused by other applications running concurrently on the cluster, this factor would explain why the first chart appears to show much better scaling than the second. We expect to continue investigating these results with the goal of isolating the impact of the various factors, but these preliminary results raise some questions about the scalability of HPC clusters for Hadoop-based data-intensive computing.

6. RELATED WORK

Power systems research most related to this work is the Frequency monitoring Network (FNET) project, under which algorithms for detecting interesting events in frequency data have been developed, including islanding, generator trips, and oscillatory behavior [16].

There is a great deal of literature on data cleaning. A formal description of data cleaning problems and methods to handle bad data can be found in [13]. Methods that target data cleaning particularly focused on sensor data can be found in [10]. Finally, an excellent book on data cleaning using exploratory data mining techniques can be found in [5].

While there are many R solutions for parallel processing, the field of big data with R is relatively new. RHadoop is a mature product from Revolution Analytics that uses R with Hadoop Streaming [14] and provides functionality very similar to RHIPE. We use RHIPE in our analyses because it is more integrated with Hadoop (being implemented in Java Hadoop vs. Hadoop Streaming) – providing more flexibility. Its data serialization format is also more compatible with non-R applications, which our work is integrating with.

7. CONCLUSIONS

The D&R methodology and computation environment enabled us to perform ad hoc exploratory analysis with a large set of historical data. We were able to discover various issues with the data and build precise algorithms to account for the erroneous activity. After accounting for the bad data, we were able capture behaviors of interest in the data and build event detection algorithms for them. All of this activity was done effectively due to our ability to look at the data in great detail and compute over the entire data set.

Many of the data quality issues we uncovered in the data would not have come to light if we had only looked at a subset of the data, or had we only looked at high-level aggregates of the data. Our use of the D&R tools to perform the iterative process of initially analyzing the entire data set to identify interesting data subsets, investigating those smaller data sets in detail to develop detection algorithms, then applying those algorithms across the entire data set was highly effective.

It is important to note how unique each type of bad data is. Simplistic data cleaning techniques, such as thresholding, would not have been able to identify all of the bad data elements we encountered. Similarly, performing analysis over a random sampling of the data would have missed many of the patterns that defined the erroneous records.

Out iterative analysis approach would have been much more difficult to achieve without the ability to quickly develop, adapt, and deploy code in a distributed fashion. The ability to not only generate and adapt code for a new task in a matter of minutes, but also to easily deploy it over several compute nodes and receive results in tens of minutes is essential to achieving an iterative analysis process.

The analyses here have only focused on one variable, frequency. Future analyses will look at other variables, particularly phase angles, as it is thought that phase angles hold a great deal of information about the stability of the power grid.

8. ACKNOWLEDGMENTS

This work was supported by the Pacific Northwest National Laboratory Future Power Grid Initiative, and was performed using PNNL Institutional Computing.

9. REFERENCES

[1] B. Akyol, J. Haack, B. Carpenter, S. Ciraci, M. Vlachopoulou, and C. Tews. Volttron: An agent execution platform for the electric power system. In *Third International Workshop on Agent Technologies for Energy Systems Valencia, Spain*, 2012.

[2] D. B. Carr, R. J. Littlefield, W. Nicholson, and J. Littlefield. Scatterplot matrix techniques for large n. *Journal of the American Statistical Association*, 82(398):424–436, 1987.

[3] W. S. Cleveland and S. J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403):596–610, 1988.

[4] Comprehensive R Archive Network. http://cran.r-project.org/, 2013. Accessed: 2013-04-10.

[5] T. Dasu and T. Johnson. *Exploratory data mining and data cleaning*. Wiley-IEEE, 2003.

[6] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] S. Guha, R. Hafen, J. Rounds, J. Xia, J. Li, B. Xi, and W. S. Cleveland. Large complex data: divide and recombine (d&r) with rhipe. *Stat*, 1(1):53–67, 2012.

[8] Hadoop. http://hadoop.apache.org, 2013. Accessed: 2014-04-04.

[9] R. Hafen. "datadr" github documentation. http://hafen.github.io/datadr/. Accessed: 2013-08-20.

[10] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. A pipelined framework for online cleaning of sensor data streams. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 140–140. IEEE, 2006.

[11] A. Kleiner, A. Talwalkar, P. Sarkar, and M. I. Jordan. A scalable bootstrap for massive data. *arXiv preprint arXiv:1112.5016*, 2011.

[12] G. M. Ljung and G. E. P. Box. On a measure of lack of fit in time series models. *Biometrika*, 65(2):297–303, 1978.

[13] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.

[14] Revolution Analytics. http://www.revolutionanalytics.com, 2013. Accessed: 2013-04-10.

[15] RHIPE. http://www.rhipe.org, 2013. Accessed: 2013-04-04.

[16] Y. Zhang, P. Markham, T. Xia, L. Chen, Y. Ye, Z. Wu, Z. Yuan, L. Wang, J. Bank, J. Burgett, et al. Wide-area frequency monitoring network (fnet) architecture and applications. *Smart Grid, IEEE Transactions on*, 1(2):159–167, 2010.