Towards Energy-Proportional Anomaly Detection in the Smart Grid

Spencer Drakontaidis, Michael Stanchi, Gabriel Glazer, Jason Hussey, Aaron St. Leger* and Suzanne J. Matthews*
Department of Electrical Engineering & Computer Science
United States Military Academy, West Point, NY 10996
Emails: [spencer.drakontaidis, michael.stanchi, gabriel.glazer, jason.hussey, aaron.stleger, suzanne.matthews]@usma.edu
*Corresponding Authors

***Abstract*—Phasor Measurement Unit (PMU) deployment is increasing throughout national power grids in an effort to improve operator situational awareness of rapid oscillations and other fluctuations that could indicate a future disruption of service. However, the quantity of data produced by PMU deployment makes real-time analysis extremely challenging, causing grid designers to invest in large centralized analysis systems that consume significant amounts of energy. In this paper, we argue for a more energy-proportional approach to anomaly detection, and advocate for a decentralized, heterogeneous architecture to keep computational load at acceptable levels for lower-energy chipsets. Our results demonstrate how anomalies can be detected at real-time speeds using single board computers for on-line analysis, and in minutes when running off-line historical analysis using a multicore server running Apache Spark.**

I. Introduction

Smart grid technologies and applications aim to increase reliability, resilience and efficiency of power grids. Smart grid technologies differentiate themselves from legacy technologies by including embedded processing, two-way communication and control capability. Phasor Measurement Units (PMUs) are one example. PMUs provide direct measurement of time synchronized voltage and current phasors throughout the power grid at sample rates significantly higher than traditional sensors. PMUs can enhance power system operations through Wide Area Monitoring and Control (WAMC) applications [1], [2], [3], [4]. WAMC applications must acquire and analyze PMU data in real-time, or near real-time. Historical analysis of PMU data can yield further insight into power system operation over time and assist in the development of real-time WAMC applications.

For example, detection and identification of power and frequency oscillations is one of the most beneficial uses of PMU data according to the North American Electric Reliability Corporation [2]. Detecting oscillations (and other anomalies in power grids) in real-time can enhance power system operator situational awareness and be integrated into automated wide-area control loops to improve WAMC capability and resiliency. However, as PMU deployment increases, communicating and processing data in real-time becomes challenging. The dataset for historical analysis grows considerably as more PMUs are installed. Both real-time and historical analysis of PMU data are investigated in this work.

The quantity of PMU data (and the computing resources required) varies widely based on the analysis being performed (real-time or historical) and the computing architectures chosen. Generally speaking, WAMC systems follow a centralized approach for implementing anomaly detection. Centralized anomaly detection requires significant communication network capacity to transfer all PMU data and typically employ cluster computing for data analysis. However, clusters are expensive to power, cool and maintain. Furthermore, they may be sub-optimal for all types of analysis. This lack of energy-efficiency becomes a bigger problem as power becomes a dominant concern in the design of computing systems.

This work investigates progress toward achieving energy-proportional anomaly detection using PMU data in smart grids. We posit this can be achieved by exploring computing techniques and architectures that reduce the total energy consumed by the system, while still achieving run-times acceptable for real-time and off-line analyses. Particularly, to improve energy-proportionality of anomaly detection in the smart grid, we focus on a.) real-time anomaly detection using single board computers and b.) off-line historical analysis of PMU data using a multicore server.

Our proposed approach to real-time anomaly detection is integrated and tested in a smart grid testbed. Our historical analysis approach is tested on 100 million real measurements produced by the testbed. Our results show that scaling the architecture to match the amount of work being performed is a successful strategy. We are able to achieve real-time performance when performing on-line anomaly detection of real PMU data. Our off-line historical analysis of 100 million real PMU measurements can be accomplished in 3.38 minutes on a single server running Apache Spark [5]. Our results support a distributed strategy of incorporating PMUs and servers throughout the grid. This prevents the transmission and accumulation of too much data at a central node, a big cause of latency and inefficiencies in current designs.

The rest of the paper is organized as follows. In section II background and related work is discussed. An overview of the proposed system and architecture is provided in section III. Section IV presents details on historical and real-time anomaly detection. Results are shown in section V and followed by concluding remarks.

II. Background

The term "energy-proportional computing" was coined by Google researchers in their seminal paper [6] observing that peak energy efficiency occurs at peak CPU utilization. The authors argue that servers should be redesigned to consume power in proportion to work being done. The ramifications of this observation increase as power consumption becomes a forcing function in the architectural design choices of all computing systems. In the context of WAMC applications, the expense and power consumption of large-scale clusters for anomaly detection obligates the exploration of more energy-efficient solutions.

Redesigning servers to be energy-proportional is difficult, and is an area under active research. However, if peak energy efficiency occurs at peak CPU utilization, we posit that we can get closer to achieving energy-proportionality in the smart grid by a.) redistributing computational load so more work is done on a single CPU and b.) using weaker, yet more energy-efficient, chipsets for analysis. The former can be achieved by leveraging shared-memory strategies for multicore execution, while the latter can be achieved by employing single board computers.

Unlike microcontrollers, single board computers (or SBCs) are fully functioning computers with much more memory and computational power. The have a plethora of ports that facilitate their integration with existing hardware. For example, the Raspberry Pi [7] has a quad-core ARM processor, 1 GB of RAM and costs only $35.00.

A. Related Work

While APIs [8], [9] for shared-memory programming have existed for decades, the learning curve required for successful deployment has caused many researchers to shy away from the explicit parallel programming required to harness multi-core systems. Unsurprisingly, researchers that explore parallel solutions for the smart grid typically focus on implicit, cluster-based solutions like Apache Hadoop [10], the most popular open source implementation of MapReduce. The MapReduce paradigm [11] takes an implicit, data parallel approach to analyzing large amounts of data. Programmers need only define a `map()` and a `reduce()` function; the underlying framework automates the rest.

Several researchers [12], [13], [14], [15], [16] have explored the use of Hadoop for smart grid applications. Most notably, the Tennessee Valley Authority (TVA) used a Hadoop cluster with 180 processing cores for historic analysis of 25 TB of PMU data collected over a period of four years [12]. Researchers in the United Kingdom [15] used Hadoop on an 8-node cluster to parallelize the analysis of 16 million simulated PMU measurements, achieving a run-time of approximately 8 minutes. Other researchers [13], [14] attempted to use Hadoop clusters for smart grid analysis with limited success, owing largely to the bottleneck of writing intermediates to the Hadoop File System (HFS) on smaller datasets.

Researchers at West Point [17] opted to use Phoenix++ [18], a shared-memory implementation of MapReduce, to detect power grid anomalies using an multicore server. They were able to detect anomalies on a large dataset of 18 million real PMU measurements in just seconds. Their system detected *constraint* anomalies (where measurements fall outside an allowed window of variation) and *temporal* anomalies (a rapid fluctuation in measured data within a window). The latter was detected using the Fano Factor, which is defined as:

$$F = \frac{\sigma_w^2}{\mu_w}$$

where σ_W^2 is the variance of measurements in the window w, and μ_w is the mean of the data in the window. Experiments showed the effectiveness [17] of the Fano factor for detecting temporal anomalies.

However, there are limitations to adopting Phoenix++ for WAMC applications. Specifically, the Phoenix++ platform is no longer under active development, and appears to have some incompatibilities with the default libraries in recent Linux versions. This suggests that Phoenix++ is not a sustainable choice for future large-scale smart grid applications.

One promising platform for smart grid applications is Apache Spark [5]. A key advantage of Apache Spark over Hadoop is its use of Resilient Distributed Datasets (RDDs) and lazy evaluation, which can significantly reduce the number of operations performed on large datasets. Like Phoenix++, Apache Spark evaluates intermediate values in memory. Both of these features can result in significant improvements in application performance. Some benchmarks suggest that Apache Spark is up to 100 times faster than Hadoop [5].

The frequency monitoring network known as FNET/GridEye [1], [19], [20], [21] is one of the earliest adopters of using Apache Spark for detecting alarm events in the Smart Grid. The authors advocate the use of Frequency Disturbance Recorders (FDRs) over PMUs, due to higher deployment cost of PMUs. The recent FNET architecture [1] uses cluster computing with Apache Spark, Hadoop and R for near real-time FDR data analysis and off-line FDR data analysis. For real-time applications, a single server which concentrates all FDR data is used. Our work focuses on alarm detection of PMU data. We note that while both FDRs and PMUs provide highly-accurate, GPS time-stamped, phasor and frequency measurements in a distributed fashion, PMUs are becoming increasingly inexpensive, provide more measurements than FDRs, and provide different functionality by monitoring the power grid at the transmission level.

Researchers recently demonstrated a frequency estimation technique using FPGAs [22]. They posit that detecting numerous other conditions to include fault detection would also be possible, which is part of what we present in this publication. Other work involving FPGAs and the smart grid have involved voltage control [23] and regulation [24] in communication control systems, but not specifically anomaly detection. Researchers also explored the use of the Raspberry Pi as a "network tap" for monitoring a testbed for network intrusions [25]. However, the Raspberry Pi appears to be used for data collection only, and not anomaly detection.

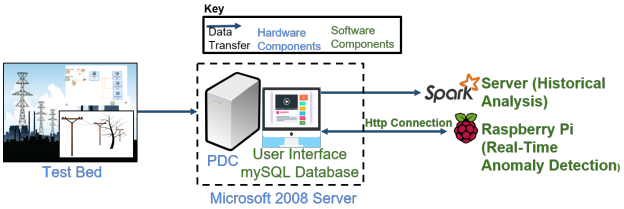

Fig. 1. System Overview

More recently, researchers have discussed the use of Raspberry Pi clusters for centralized anomaly detection [26], and proposed the use of Raspberry Pis in a distributed manner for real-time anomaly detection [27]. In the latter paper, a simulation suggested that a single Raspberry Pi was capable of detecting anomalies in up to 50 PMUs at real-time speeds [27]. The work presented in this paper implements the strategy mentioned in prior work in a smart grid testbed, confirming the feasibility of the design.

B. Our Contributions

In this paper, we present an energy-proportional strategy to anomaly detection in the smart grid. Specifically, we:

- Integrate a Raspberry Pi based real-time anomaly detector (on-line detection)
- Design a parallel, multicore Apache Spark approach for historical anomaly detection (off-line detection)

Our work has several advantages over prior work. To the best of our knowledge, we are the first to integrate a Raspberry Pi for real-time anomaly detection in the smart grid. Not only are we able to achieve real-time on-line analysis, but we are able to do on a device that has a peak power draw of 4 Watts. Likewise, we believe we are the first to show that historical analysis for anomalies is possible using a multicore instance of Apache Spark. Unlike prior work which focuses on Apache Spark *clusters*, we are able to detect anomalies in minutes on a single multi-core node. We also benchmark our historical analysis on a large, real dataset consisting of over 100 million PMU measurements derived from a smart grid testbed. Our 36-core Apache Spark server under peak load consumes just 330 Watts.

Our results clearly show that we are able to drastically cut down the power requirements required for smart grid analysis, and make critical progress toward creating a more energy-proportional anomaly detection system for the smart grid.

III. System Overview

Figure 1 gives an overview of our smart grid architecture. The system is comprised of a physical testbed; a Microsoft 2008 server that houses the Phasor Data Concentrator (PDC) and a web-based user interface that allows a user to interact with the grid; a connection to an Apache Spark server for historical analysis; and a connection to a Raspberry Pi for real-time analysis.

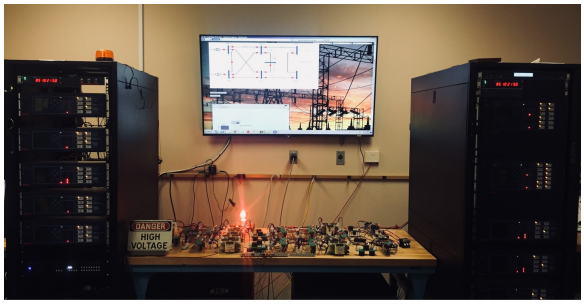

Fig. 2. USMA Smart Grid Testbed

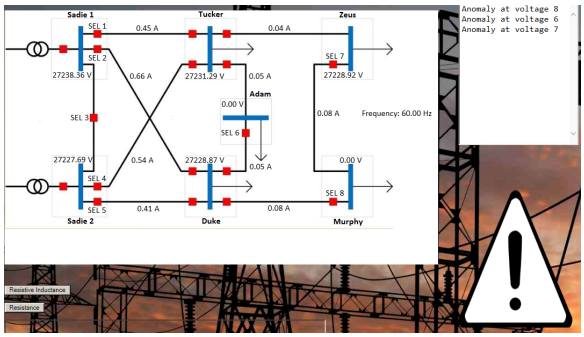

Fig. 3. User Interface with Anomaly Notification

a) The physical testbed: Our smart grid testbed (Figure 2) is a 1:1000 scaled version of a 46kV power system with nine transmission lines, seven-buses, eight real Phasor Measurement Units (PMUs), solar micro-inverters to simulate power generated from solar panels, a controllable load to alter the resistance and resistive inductance the grid experiences, a fault generator to introduce anomalies to the grid, and a static VAR compensator [28]. The PMUs measure voltage and current at each bus in the testbed, calculate voltage and current phasors, system frequency, and other derived quantifies as outlined in the IEEE synchrophasor standard [29]. The PMUs send data via TCP/IP to our server in accordance with the IEEE synchrophasor standard [30].

b) The PDC/UI server: Both the Phasor Data Concentrator (openPDC) and MySQL database are currently housed on a Windows 2008 server. The server collects PMU data from the testbed and runs it through openPDC, which time-aligns the data and stores them in a MySQL database. OpenPDC assigns each measurement a signal ID. The signal ID is a 32-character hexadecimal hash that uniquely identifies the associated PMU and the type of measurement. We use the signal ID throughout the detection process to identify each measurement's allowed levels of variation. The data stored in the MySQL database can be queried later for either historical or real time analysis.

This server also hosts the user interface (UI), a website that integrates the hardware modules and the anomaly detection software. The UI consists of static HTML and an AJAX (Asynchronous JavaScript and XML) framework to display grid readings and anomaly notifications in real-time (see Figure 3). Our UI also integrates the jQuery module Highcharts [31] to

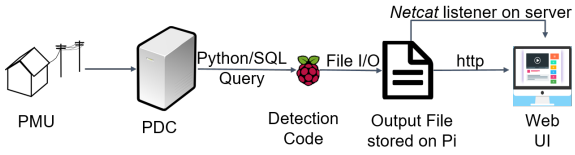

Fig. 4. Overview of Real-Time Analysis

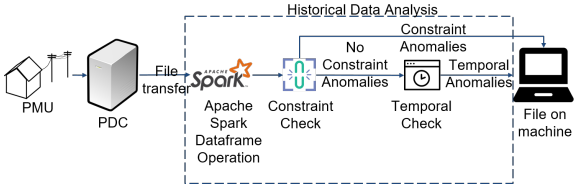

Fig. 5. Overview of Historical Analysis

enable grid operators to visualize and "zoom in" on variable time intervals from historical data.

c) Raspberry Pi for Real-time Analysis: The Pi is connected to the Windows 2008 server over Ethernet in a closed, private network. This allows for speed on network communications and reduces the attack surface to help provide security for the system as a whole. The Pi analyzes the data coming in from the grid, and sends results back to the server which displays them on the UI (Figure 3). The Pi sends a query to the database in half-second intervals for anomaly detection. Detected anomalies are relayed back to the server using the Python `requests` library. The UI checks for new anomalies every half second and displays them to the user.

d) Apache Spark server for Historical Analysis: For historical analysis, we utilize a separate server running Apache Spark. Data is queried from the MySQL database and is aggregated into a CSV file. This CSV file is compressed and sent to the Apache Spark server for off-line analysis. Unlike on-line analysis, off-line analysis is expected to be executed periodically. In our case, we perform off-line analysis on data collected over 12 hours from the smart grid testbed.

IV. Anomaly Detection

The input to the anomaly detection approach is a comma-separated value (CSV) file produced by a database query containing all recorded measurements from a particular interval, and a CSV file containing operator-determined constraints for each PMU. Each line of the input file contains a single measurement, consisting of a signal ID, a time stamp, and the value of the measurement. Recall that the signal ID is a hash that uniquely identifies a specific type of measurement coming from a certain PMU. For our 1-phase distribution grid, the 8 PMUs measure voltage and current phasors, system frequency, and change in frequency every 16.67 ms, corresponding to a reporting rate of 60 Hz. Thus, while reporting single phase data with 8 PMUs, 40 measurements are recorded every 16.67 ms. We refer to a 40-measurement block of data as a *data frame*. Each line of the operator-defined constraint file consists of a a signal ID, PMU number, measurement type, minimum and maximum values defining the window of allowed variation, and the maximum allowed Fano factor limit. The constraint data is indexed by signal ID and loaded into a global hash table prior to analysis.

A. Real-Time Analysis

Figure 4 illustrates the query - analyze - alert cycle executed by our Raspberry Pi during real-time analysis. This aspect of the system represents the first real-world implementation of the distributed Pi anomaly detection system proposed in prior work [27]. We employ a Python wrapper around a core C program [27], owing to the lack of C libraries for querying MySQL databases and the relative speed of C over Python. For real-time analysis, the goal is to process a frame of data as quickly as data is produced (16.67 milliseconds).

The Python program on the Raspberry Pi first queries data directly from the MySQL database via the Python `mysql` module. The query finds the 1200 most recent measurements in the database (determined by the time stamp stored with each measurement). We note here that 1200 measurements is equivalent to 30 frames of data which, at a 60 HZ reporting rate, is produced every 0.50 seconds. Thus querying 1200 measurements every half-second ensures we keep pace with the real-time production of measurements from the grid. Another advantage of the 1200 measurement interval is that it lowers the overheard cost associated with querying the database. We note that the user can decide how big each aggregation interval, and retains the ability to slow down or speed up the collection rate of the script as needed.

The Python program uses the `os.subprocess` module to call a C program that reads in the data, processes it for anomalies, and outputs any anomalies to a file on the Pi. The Python program reads in any detected anomalies, assigns them identification numbers, and converts them into JSON format, where the identification number is used as the key and the entire line of data is used as the value. This JSON object is then sent to a dedicated HTTP server on the Windows Server that receives this data via an HTTP POST request.

The HTTP Server listens for POST requests, feeding received data into a dictionary, which aggregates the data into human operator readable form. Each individual POST request is logged so that the data can be accessed at a later time. Once the aggregation interval is complete, the HTTP Server writes the data to a file that is consumed by a JavaScript function every half second. The GUI is then updated with any anomalies that were detected.

B. Historical Analysis

Figure 5 gives an overview of our historical analysis process. Unlike the previous work [17], where constraint and temporal anomalies were analyzed over two iterations of MapReduce in Phoenix++, we design a new MapReduce algorithm that enables anomaly detection using only a single MapReduce instance in Apache Spark. Furthermore, unlike prior work [17], the principle purpose of MapReduce here

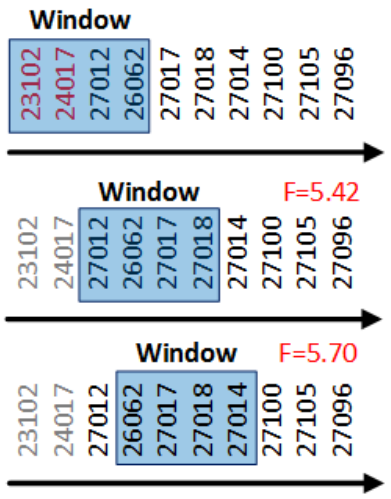

Fig. 6. Anomaly Detection Scheme: Reduce Phase

is to sort the data by signal IDs. Anomaly detection occurs entirely in the reduce phase.

During the Map phase, input data is split between each instance of the `map()` function (or *mapper*), and creates *(key,value)* pairs where each signal ID is the key, and the value is a tuple consisting of time-stamp and associated measurement. Note that each mapper executes independently and in parallel. The outputted *(key,value)* pairs are fed to a Combiner, which hashes all common signal IDs together to create individual queues for each signal ID. At the conclusion of the Map and Combiner phases, the data is organized by signal ID into (signal ID, list(measurements)) pairs.

During the Reduce phase, each instance of the `reduce()` function (or *reducer*) receives a set of queues and examines each for anomalies. Prior to analysis, each queue's data is sorted according to time-stamp. Next, we run a sliding window across all the measurement data. The width of the window corresponds to the reporting rate. Our testbed has a 60 Hz reporting rate. Thus, we set the window size to 60 measurements. We note the window size can be adjusted to fine tune the performance of anomaly detection.

To illustrate this process, we present an example of voltage anomaly detection in Figure 6. Prior to the start of the process, we do a hash table lookup on the signal ID and retrieve the measurements that define the window of allowed variation for voltage. In this case, the nominal range for voltage measurements is anything between $25,000$ V and $27,600$ V. Any measurement outside this window is labeled a constraint anomaly. We also look up the Fano Factor limit associated with the signal ID (in this case, experimentally derived [17] as 2.7). In this example, we use a window size of 4.

The anomaly detection process begins with a constraint check on the first window of measurements, which are the first four measurements listed in Figure 6. In the first window, the first two elements follow below the allowed range for voltage. Thus, these two measurements are returned as constraint anomalies. Measurements three and four in contrast are not marked as constraint anomalies, as they fall within the nominal range. Thus, the start position of the window is advanced to measurement three, or $27,012$ V (Figure 6). The constraint check is repeated on all elements in this window. Since there are no constraint anomalies present in the window, we next check the window for temporal anomalies.

During temporal analysis, the Fano factor of the measurement data within the window is calculated. In the case of the example presented in Figure 6, the calculated Fano factor for the window defined by the measurements $27,012 \ldots 27,018$ is 5.42, which is above our Fano factor limit. Thus, this window is marked as having a temporal anomaly, and the Fano factor along with the starting and ending timestamps is outputted to the user. The window's position is then advanced by one. In Figure 6, the new window is now defined by $26,062 \ldots 27,014$. We first perform a constraint check on the new measurement inserted into the window $(27,014)$. Since this is within the nominal range, we recompute the Fano factor, and find that is now 5.70 which is again anomalous. This window's timestamps are also outputted to the user.

The window's position is advanced by one each time, with a constraint check performed on the newest measurement added to the window. If a constraint anomaly is found, the window's start position is advanced one past the anomalous measurement. Otherwise, temporal anomaly checks are performed on the sliding window. This process repeats until the measurement list associated with the signal ID is exhausted. In Figure 6, the computed Fano factors for each of the subsequent windows are below the Fano factor limit, so no additional windows are output as being anomalous.

V. Experimental Benchmarking & Results

Benchmarking was performed on real PMU measurement data derived from the USMA testbed. For real-time anomaly detection, the Raspberry Pi reads measurements as the grid testbed operates. For historical analysis, we recorded 100 million measurements over a 12-hour period, corresponding to approximately 7.4 GB of data.

Historical Analysis experimentation was run on DOD supercomputer Topaz, a SGI ICE X system located at the U.S. Army Engineer Research & Development center. The supercomputer is comprised of $3,456$ nodes, each with a 36-core Intel Xeon E5-2699v3 Haswell processor, with core speeds of 2.3 GHz and 128 GB of Random Access Memory. Each node consumes approximately 330 Watts of power at peak load. Since our goal was to achieve efficient speeds using as few resources as possible, experimentation was conducted on a single compute node running Apache Spark.

Real-time experimentation was run on a Raspberry Pi 3 Model B single board computer, which retails for $35.00. The Raspberry Pi 3 features a quad-core ARM Cortex-A53 processor at 1.2 GHz, and 1 GB of RAM. Recent benchmarks [32], [33] show that the Raspberry Pi 3 consumes up to 4 Watts of power at peak load.

TABLE I
REAL-TIME ANOMALY DETECTION TIMES

Raspberry Pi Detection Workflow Component	**Average Time (ms)**	**Real-time (ms)**
Query Time (Latency)	5.3	NA
Results Notification to Server Time (Latency)	25	NA
Anomaly Detection Time (30 Data frames)	**20**	500
Anomaly Detection Time (1 Data frame)	**0.40**	16.67

TABLE II
HISTORICAL ANOMALY DETECTION TIMES (S)

Step	10 Million	50 Million	100 Million
Initialize Spark	1.75	1.76	1.75
Read In Data	6.74	6.69	6.74
Anomaly Detection	24.49	95.81	194.69
Total Time	32.98	104.26	203.18

a) Real Time Analysis Results: Table I illustrates our real-time benchmarking results. Time is measured in milliseconds (ms), and illustrates the amount of time required for our on-line anomaly detection workflow as compared to timing requirements for real-time analysis. Specifically, data must be processed for anomalies at a faster rate than data is produced. For the anomaly detector reporting every 2 Hz and PMUs reporting at 60 Hz, we must analyze this data in 500 ms (or 16.67 ms per data frame). Average anomaly detection time represents the average of five runs.

The time required to query the database for 1200 measurements and send the results back to the Windows 2008 server represent the *latency* of our system, and averages approximately 30.30 ms for 30 frames of data. Next, the time to detect all anomalies in our 1200 measurements and output them to a file takes approximately 20 ms. Thus, the total anomaly detection time takes 50.30 ms, well below the 500 ms limit for required for real-time analysis. For a single data frame, the average time is 0.40 ms. Our results clearly indicate that we meet the standard for real-time anomaly detection. Lastly, CPU utilization on the Pi stayed steady between 66% and 71%. This is perhaps unsurprising, as prior simulation results [27] suggested that a Raspberry Pi could support up to 50 PMUs and still meet the definition of real-time.

b) Historical Analysis Results: Table II shows our raw run times for off-line historical anomaly detection. For these tests, we vary the input data set size from 10 million to 100 million measurements by increments of 10 million. For each data size, we run the algorithm five times.

Spark initialization is consistent over our different datasets, averaging approximately 1.75 seconds. The Anomaly detection time includes the time to detect anomalies along with the amount of time required to format the data in a manner that can be outputted to the user. We note that file write time is not included, as this time is highly variable and dependent on the number of detected anomalies. We find that the anomaly detection time is roughly linear with the size of the input. For 100 million measurements, we are able to detect all anomalies in 3.38 minutes. CPU utilization after startup for the 100 million dataset varied between 52% to 73%, suggesting that the system would support analysis of larger files, or a smaller server could be used.

VI. CONCLUSION

Designing compute systems that consume energy in proportion to the amount of work performed becomes a critical goal as energy consumption becomes the dominant factor in system design. In this paper, we propose a system to better achieve "energy-proportional" anomaly detection in the smart grid. Instead of proposing a new server design, we suggest a two-prong approach that shifts anomaly detection to processors that better match the computational load required of analysis. Our goal here is not to get the *fastest* results, but *acceptable* results while maintaining relatively low power consumption. To the best of our knowledge, we are the first to suggest such a strategy for anomaly detection in the smart grid.

For real-time analysis, we are the first to integrate a Raspberry Pi into a smart grid testbed for on-line anomaly detection. Our results indicate that the Pi is able to analyze data in real-time while requiring at most a 4 Watt power draw. This lower consumption coupled with the inexpensive nature of the Pi support the notion that single board computers like the Pi can be used to enable real-time anomaly detection of PMU data. In a larger system, Pis would be networked to a subset of PMUs throughout the grid.

For off-line historical analysis, we show Apache Spark executing on a single node is capable of analyzing large amounts of data very efficiently. Like real-time analysis, we believe that historical analysis should be distributed throughout the grid, with historical analysis concentrated on local power grids. We also believe a more "edge-computing" approach to on-line and off-line anomaly detection prevents data from snowballing into a big data problem that would require the large number of computational resources that necessitate clusters.

Our results strongly support our strategy for creating a more energy-efficient smart grid. By shifting computational load to processors that can still analyze the data at acceptable rates, we are maximizing the energy-efficiency of our system. We believe that our work represents an important step toward creating a more energy-efficient smart grid, and will also be of high interest to anyone trying to improve the energy efficiency of alarm analysis in other ICS/SCADA systems.

ACKNOWLEDGMENT

Funding for this project was provided by the Office of Naval Research, the U.S. Army Armament Research, Development and Engineering Center (ARDEC) and the DOD High Performance Computing Modernization Program (HPCMP). We are especially grateful to the HPCMP program for a grant of time on the supercomputer Topaz. The views expressed in this article are those of the author and do not reflect the official policy or position of the Department of the Army, Department of Defense or the U.S. Government.

REFERENCES

[1] D. Zhou, J. Guo, Y. Zhang, J. Chai, H. Liu, Y. Liu, C. Huang, X. Gui, and Y. Liu, "Distributed data analytics platform for wide-area synchrophasor measurement systems," *IEEE Transactions on Smart Grid*, vol. 7, no. 5, pp. 2397–2405, Sept 2016.

[2] NERC, "Real-time application of synchrophasors for improving reliability," *Tech. Rep. 1*, 2010.

[3] A. St. Leger, J. James, and D. Frederick, "Smart grid modeling approach for wide area control applications," in *2012 IEEE Power and Energy Society General Meeting*, July 2012, pp. 1–5.

[4] K. M. Koellner, S. Burks, B. Blevins, S. N. Nuthalapati, S. Rajagopalan, and M. L. Holloway, "Synchrophasors across texas: The deployment of phasor measurement technology in the ercot region," *IEEE Power and Energy Magazine*, vol. 13, no. 5, pp. 36–40, Sept 2015.

[5] Apache Foundation, "Apache Spark: Lightning-fast cluster computing," Internet Website, 2013. [Online]. Available: http://spark.apache.org/

[6] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, 2007.

[7] Raspberry Pi Foundation. (2016) Raspberry pi 3 model b. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[8] B. Nichols, D. Buttlar, J. Farrell, and J. Farrell, *Pthreads programming: A POSIX standard for better multiprocessing*. " O'Reilly Media, Inc.", 1996.

[9] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[10] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[11] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[12] P. Trachian, "Machine learning and windowed subsecond event detection on pmu data via hadoop and the openpdc," in *IEEE PES General Meeting*. IEEE, 2010, pp. 1–5.

[13] M. Edwards, A. Rambani, Y. Zhu, and M. Musavi, "Design of hadoop-based framework for analytics of large synchrophasor datasets," *Procedia Computer Science*, vol. 12, pp. 254–258, 2012.

[14] F. Bach, H. K. Çakmak, H. Maass, and U. Kuehnapfel, "Power grid time series data analysis with pig on a hadoop cluster compared to multi core systems," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE, 2013, pp. 208–212.

[15] M. Khan, P. M. Ashton, M. Li, G. A. Taylor, I. Pisica, and J. Liu, "Parallel detrended fluctuation analysis for fast event detection on massive pmu data," *IEEE Transactions on Smart Grid*, vol. 6, no. 1, pp. 360–368, 2015.

[16] V. Fanibhare and V. Dahake, "Smartgrids: Mapreduce framework using hadoop," in *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*, Feb 2016, pp. 400–405.

[17] S. J. Matthews and A. St. Leger, "Leveraging mapreduce and synchrophasors for real-time anomaly detection in the smart grid," *IEEE Transactions on Emerging Topics in Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[18] J. Talbot, R. M. Yoo, and C. Kozyrakis, "Phoenix++: Modular mapreduce for shared-memory systems," in *Proceedings of the Second International Workshop on MapReduce and Its Applications*, ser. MapReduce '11. New York, NY, USA: ACM, 2011, pp. 9–16. [Online]. Available: http://doi.acm.org/10.1145/1996092.1996095

[19] J. Chai, Y. Liu, J. Guo, L. Wu, D. Zhou, W. Yao, Y. Liu, T. King, J. R. Gracia, and M. Patel, "Wide-area measurement data analytics using fnet/grideye: A review," in *2016 Power Systems Computation Conference (PSCC)*, June 2016, pp. 1–6.

[20] Y. Liu, W. Yao, D. Zhou, L. Wu, S. You, H. Liu, L. Zhan, J. Zhao, H. Lu, W. Gao, and Y. Liu, "Recent developments of fnet/grideye: A situational awareness tool for smart grid," *CSEE Journal of Power and Energy Systems*, vol. 2, no. 3, pp. 19–27, Sept 2016.

[21] Y. Liu, L. Zhan, Y. Zhang, P. N. Markham, D. Zhou, J. Guo, Y. Lei, G. Kou, W. Yao, J. Chai, and Y. Liu, "Wide-area-measurement system development at the distribution level: An fnet/grideye example," *IEEE Transactions on Power Delivery*, vol. 31, no. 2, pp. 721–731, April 2016.

[22] E. F. C. Grabovski and S. A. Mussa, "Three-phase frequency estimator in smart grid applications: Practical issues using fpga," in *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, June 2017, pp. 175–179.

[23] L. Miao, G. Wei, X. Fang, and J. Risheng, "The strategy of the voltage control in smart grid based on modern control method and fpga," in *2015 34th Chinese Control Conference (CCC)*, July 2015, pp. 8964–8968.

[24] N. Nila-Olmedo, F. Mendoza-Mondragon, A. Espinosa-Calderon, and Moreno, "Arm+fpga platform to manage solid-state-smart transformer in smart grid application," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Nov 2016, pp. 1–6.

[25] G. Koutsandria, R. Gentz, M. Jamei, A. Scaglione, S. Peisert, and C. McParland, "A real-time testbed environment for cyber-physical security on the power grid," in *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy*, ser. CPS-SPC '15. New York, NY, USA: ACM, 2015, pp. 67–78. [Online]. Available: http://doi.acm.org/10.1145/2808705.2808707

[26] K. Candelario, C. Booth, A. S. Leger, and S. J. Matthews, "Investigating a raspberry pi cluster for detecting anomalies in the smart grid," in *2017 IEEE MIT Undergraduate Research Technology Conference (URTC)*, Nov 2017, pp. 1–4.

[27] S. J. Matthews and A. St. Leger, "Leveraging single board computers for anomaly detection in the smart grid," in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, Oct 2017, pp. 437–443.

[28] A. St. Leger, J. Spruce, T. Banwell, and M. Collins, "Smart grid testbed for wide-area monitoring and control systems," in *2016 IEEE/PES Transmission and Distribution Conference and Exposition (T D)*, May 2016, pp. 1–5.

[29] "Ieee standard for synchrophasor measurements for power systems," *IEEE Std C37.118.1-2011 (Revision of IEEE Std C37.118-2005)*, pp. 1–61, Dec 2011.

[30] "IEEE standard for synchrophasor data transfer for power systems," *IEEE Std C37.118.2-2011 (Revision of IEEE Std C37.118-2005)*, pp. 1–53, Dec 2011.

[31] Highcharts. (2017) Highcharts api. [Online]. Available: https://api.highcharts.com/highcharts/

[32] J. Geerling. (2018) Raspberry pi dramble: Power consumption benchmarks. [Online]. Available: https://www.pidramble.com/wiki/benchmarks/power-consumption

[33] MagPi:the official Raspberry Pi magazine. (2018) Raspberry pi 3b+ specs and benchmarks. [Online]. Available: https://www.raspberrypi.org/magpi/raspberry-pi-specs-benchmarks/