IoTAbench: an Internet of Things Analytics Benchmark

Martin Arlitt
HP Labs
Palo Alto, CA
martin.arlitt@hp.com

Manish Marwah
HP Labs
Palo Alto, CA
manish.marwah@hp.com

Gowtham Bellala
HP Labs
Palo Alto, CA
gowtham.bellala@hp.com

Amip Shah
HP Labs
Palo Alto, CA
amip.shah@hp.com

Jeff Healey
HP Vertica
Cambridge, MA
jeff.a.healey@hp.com

Ben Vandiver
HP Vertica
Cambridge, MA
ben.vandiver@hp.com

ABSTRACT

The commoditization of sensors and communication networks is enabling vast quantities of data to be generated by and collected from cyber-physical systems. This "Internet-of-Things" (IoT) makes possible new business opportunities, from usage-based insurance to proactive equipment maintenance. While many technology vendors now offer "Big Data" solutions, a challenge for potential customers is understanding quantitatively how these solutions will work for IoT use cases. This paper describes a benchmark toolkit called IoTAbench for IoT Big Data scenarios. This toolset facilitates repeatable testing that can be easily extended to multiple IoT use cases, including a user's specific needs, interests or dataset. We demonstrate the benchmark via a smart metering use case involving an eight-node cluster running the HP Vertica analytics platform. The use case involves generating, loading, repairing and analyzing synthetic meter readings. The intent of IoTAbench is to provide the means to perform "apples-to-apples" comparisons between different sensor data and analytics platforms. We illustrate the capabilities of IoTAbench via a large experimental study, where we store 22.8 trillion smart meter readings totaling 727 TB of data in our eight-node cluster.

1. INTRODUCTION

The commoditization of sensors and communication networks is enabling vast quantities of data to be generated by and collected from cyber-physical systems. This "Internet-of-Things" (IoT) makes possible new business opportunities, such as usage-based insurance and proactive equipment maintenance. IDC, a market intelligence firm, estimates that by 2020 there will be over 200 billion Internet-connected "things" installed [20]. For organizations wanting to pursue an IoT business, a question they are immediately faced with is how to store and analyze the vast amount of data that their sensors (i.e., "things") will collect.

ICPE'15, Jan. 31–Feb. 4, 2015, Austin, Texas, USA.

http://dx.doi.org/10.1145/2668930.2688055.

A related challenge for these organizations is understanding which Big Data analytics platform will work best for their specific needs. Currently, there are no industry standard benchmarks to aid such organizations in selecting an analytics platform. In this paper we introduce a benchmark toolkit called IoTAbench (Internet of Things Analytics benchmark). IoTAbench is envisioned as a suite of benchmarks, each of which represents a distinct IoT use case. To date we have implemented a benchmark for a smart metering use case. We envision adding other benchmarks (e.g., for building energy management, fleet management, network security, etc.) under the IoTAbench umbrella in the near future.

A key component of our smart metering benchmark is a tool for generating large volumes of synthetic sensor data with realistic properties. To demonstrate the ability of this benchmark to test a Big Data analytics platform, we use IoTAbench to evaluate the performance of the HP Vertica Analytics Platform [17] when used to manage a large IoT dataset.

Our main contributions are: (1) the description of an IoT benchmark (IoTAbench). We intend to contribute our implementation of IoTAbench to facilitate and speed the development of an industry standard IoT benchmark. (2) a realistic synthetic smart meter data generator based on an augmented Markov chain model. (3) the demonstration of IoTAbench to evaluate the performance and scalability of a commercial "Big Data" platform. (4) one of the largest Big Data research studies we are aware of (22.8 trillion readings, 727 TB of data), to help to bridge the divide between Big Data research and practice.[1] (5) a detailed IoT case study of an electric utility with 40 million smart meters (for which we store and analyze the equivalent of more than a decade of data).

The remainder of this paper is organized as follows. Section 2 describes the IoT benchmark we created. Section 3 explains our experimental methodology. Section 4 discusses the results of our experiments. Section 5 examines related work. Section 6 concludes the paper with a summary of our work and possible next steps.

[1]We use TB=terabyte= 10^{12} bytes throughout the paper.

2. IoTAbench: AN IoT BENCHMARK

The Internet of Things will encompass a broad range of workloads; we anticipate that an industry standard IoT benchmark will encompass a suite of IoT workloads covering different use cases. This "benchmark suite" approach has been adopted in other industry standard benchmarks such as SPECweb2009, which contained three different workloads (banking, e-commerce and support).[2]

We refer to our benchmark as IoTAbench, the Internet of Things Analytics benchmark. IoTAbench consists of three components: a scalable synthetic data generator; a set of SQL queries; and a test harness. To help guide the initial implementation of IoTAbench, we focused on a single use case - smart metering. The development of additional use cases is left for future work.

2.1 IoT Use Case

In this paper, we focus on one specific use case, *smart metering*. An important step in facilitating more effective use of resources (e.g., electricity, gas, water) is the deployment of more capable meters. Smart metering will enable consumers and producers alike to better understand resource usage, so that actions can be taken to eliminate inefficient use of resources or to alter consumer behavior regarding when resources are consumed. While many resource/utility providers are either considering rollouts of smart meters, in the process of deploying them, or have already installed them, there is still a lot of uncertainty surrounding smart metering. One thing that is clear is that smart metering has the potential to produce enormous volumes of data.

We focus on smart metering as a use case because it is a timely problem. For example, China is rolling out hundreds of millions of smart meters, to enable programs such as Time-of-Use (ToU) billing to raise energy awareness amongst consumers.[3] Similarly, utilities in Europe are rolling out smart meter deployments as part of initiatives to achieve climate change targets such as a 20% increase in energy efficiency across Europe by 2020.[4]

In parallel to the deployment of meters, the utilities must determine how they will store and analyze all of the collected meter data in a sustainable manner. The development of an industry standard benchmark would help utilities make more informed decisions about which analytics platform to use. This is the motivation for our study.

2.2 Synthetic Data Generator

Experimentally evaluating the performance of an analytics platform in a smart metering use case requires access to large volumes of meter data. However, the availability of empirical smart meter data at tera-scale is limited or non-existent. To facilitate testing for an electric utility case study with 40 million smart meters, a compelling option is to generate a large synthetic dataset; however, this can lead to misleading or erroneous results if the synthetic data does not have properties of the empirical data. For example, if synthetic data is generated by duplication, a much higher degree of data compression may be attained. On the other hand, if the synthetic data is completely random, data compression is likely to be poorer than for an empirical dataset. Furthermore, the time to perform analytics on the data can be affected, which would further bias the benchmark results. Thus, it is important to make the synthetic data as realistic as possible. This was our objective.

2.2.1 Generator design

We developed a Markov chain-based realistic synthetic data generator for smart meter data. The objective is to generate time series data of power consumption for any number of users, given the time series for a limited number of users, such that important statistical properties of the generated time series is similar to those of the real time series. We achieve this by simulating the power consumption process as a variant of a Markov chain.

Markov chains are widely used for modeling sequential, stochastic processes. For example, they have been used in the past for generating synthetic data for wind speed [14, 25]. A discrete-time Markov chain consists of a finite number of states, $\mathcal{S} = 1, 2, ..., n$, where state changes occur at discrete time steps; n is the number of states. The transitions between states exhibit the Markov property, that is, given the current state, the next (future) state is conditionally independent of the past states:

$$P(S_{t+1} = j | S_t = i, S_{t-1} = i_{t-1}, ..., S_0 = i_0) = P(S_{t+1} = j | S_t = i) \quad (1)$$

A state transition matrix, P of size $n \times n$, contains all the transition probabilities, where entry P_{ij} corresponds to $P(S_{t+1} = j | S_t = i)$. Furthermore, transition probabilities from a particular state must add to 1, $\sum_j P_{ij} = 1$.

We initially used a Markov chain to generate smart meter data, but the results were poor. This was not surprising, since in addition to the previous state the consumption process depends on several contextual features such as time of day, weather, etc. In order to capture the dependence on these contextual features and incorporate them into the model, we augment the Markov chain model by adding additional inputs. Figure 1 shows an Markov chain augmented with the current hour, H_t, added as an input to each state. The states of the chain are obtained by using fixed-width binning to quantize the consumption in the empirical data. With this modification, the transition to a state now depends on both the previous state and the current hour.

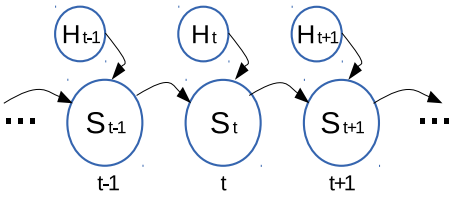


Figure 1: Augmented Markov chain of consumption states.

Using the generator involves two main steps: 1) training the model, which entails learning model parameters from a small empirical data set; and 2) performing a random walk on the augmented chain to generate the synthetic data. We describe each of these in turn.

2.2.2 Model training

Training of the model involves learning two main sets of parameters from the empirical data: 1) the transitional

[2] http://www.spec.org/web2009

[3] http://www.greentechmedia.com/articles/read/china-wants-time-of-use-pricing-by-2015-one-meter\-per-home-by-2017

[4] http://ec.europa.eu/europe2020/targets/eu-targets/index_en.htm

probability matrix (TPM), P; and 2) the probability density functions corresponding to each state. Given the additional time of day dependence, P is now $n \times n \times m$, where n is the number of states, m is the number of hours, and entry P_{ijk} corresponds to the transition probability from state i to state j at hour k; that is, the conditional probability $P(S_{t+1} = j|S_t = i, H_{t+1} = k)$. We use maximum likelihood estimation to estimate the TPM from the empirical data. In many cases, there may not be any empirical data points corresponding to a transition, and the TPM could be sparse. To address this, we use Laplace smoothing, which increases the count for each transition by one so that there are no transitions with zero probability. How sparsity is handled can be used to tune how "different" the generated data is from the empirical.

Even for a moderate n and m, the size of the TPM can be very large. For example, for $n = 50, m = 24$, its size is $n^2m(60,000)$, which is a large number of probabilities and would require a large dataset to estimate robustly. To reduce the number of probabilities, we use Bayes rule and conditional independence assumptions to factor the conditional probability.

$$\begin{aligned} &P(S_{t+1} = j|S_t = i, H_{t+1} = k) \\ \propto &P(S_t = i, H_{t+1} = k|S_{t+1} = j) \times P(S_{t+1} = j) \\ \propto &P(S_t = i|S_{t+1} = j) \times P(H_{t+1} = k|S_{t+1} = j) \times P(S_{t+1} = j) \end{aligned} \quad (2)$$

which can be normalized to determine the probabilities. This results in a large drop in the number of probabilities to be estimated. For the above example, the number decreases to $n^2 + nm + n$ (3,750 for the above example), which is a reduction of more than an order of magnitude. For each of the n states, we use a kernel density estimate on the empirical data to compute the probability density function (pdf) corresponding to that state. The estimated pdf, f, at any point x, can be expressed as

$$f_h(x) = \frac{1}{mh}\Sigma_{i=1}^{m} K(\frac{x - x_i}{h}) \quad (3)$$

where h is the selected bandwidth; m is the total number of points; K is the selected kernel; x_i are the points that fall within that state. We used a Gaussian kernel, although other kernels could also be used. If the number of points, m, is large, a binned kernel density estimate can be used. The bandwidth parameter, h, provides another knob to control the difference between the empirical and generated data. We also estimate the marginal probabilities of each state.

2.2.3 Data generation

Once the Markov chain model parameters are known, the process to generate a synthetic consumption time series involves performing a random walk on the chain. We randomly pick an initial state based on the marginal probability distribution for the starting hour. Then each subsequent state is picked based on the TPM. When a particular state is picked, we generate a consumption value by sampling the pdf of that state. To make this process a bit simpler and faster, we pre-sample a large number of points (over 100,000) from the pdf of each state and save these points. At the time of generation, we uniformly sample from these pre-saved points.

2.2.4 Implementation

Our generator is implemented as two modules. The first is the parameter learning module. We implemented it in R to estimate the model parameters from empirical data in the manner described above. R was chosen for this module since it provides a rich set of analytics packages, such as those for kernel density estimation. The speed of training is not critically important as it only needs to be done once for a particular dataset. The second module is the data generation module; efficiency is important in this step, to enable large datasets (e.g., 100's of TB) to be generated in a timely manner. For this reason, the data generation module is implemented in C. The data generation process is embarrassingly parallel. Thus, our data generation module is multi-threaded, and can be distributed across multiple machines to reduce the time needed to generate large synthetic datasets. We carefully select the random number seeds, such that each thread generates unique results compared to all other threads in the distributed generator. There is no communication between the threads, to improve the scalability of the generator.

Each row in the generator output is a meter reading. The format is:

```
timestamp meter_identifier consumption_value
```

This is the format used in the empirical dataset that we trained our model on. An example reading from the generator is:

```
2015-01-01 13:10:00|12345|103
```

This reading is for January 1, 2015 at 1:10pm from meter 12345. Over the previous 10 minutes the customer used 103 Watts of electricity.

2.2.5 Model validation

To demonstrate the effectiveness of our generator, we trained it on real smart meter data from Ireland.[5] This dataset has about 6,400 timeseries of 1.5 years each. We randomly picked 10% of the timeseries for training. The validation results are based on the entire dataset.

To capture seasonal behavior, we train a separate model for each month. Further, each day of the month is modeled as a mixture model of the current month, previous and next months. We use Gaussian weights with a tunable variance. In these experiments, we picked a variance value such that a month has a non-zero component from the middle of the previous month to the middle of the next month, as shown in Figure 2. While the variance can be varied, this choice worked well.

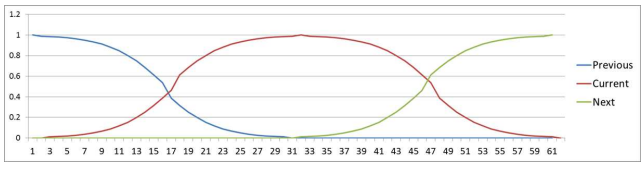
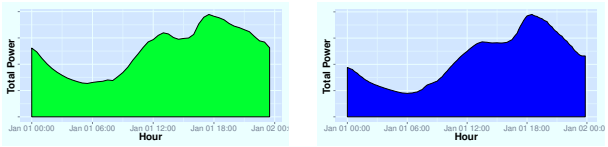


Figure 2: Mixture model weights of the three components over a 60 day period.

[5] `http://www.ucd.ie/issda/data/commissionforenergyregulationcer/`

To evaluate the quality of the generated data, we compared the empirical and generated data by looking at their marginal and conditional (by hour, month) probability distributions and the auto-correlation within each time series. All of these matched closely. Figure 3 shows the empirical and synthetic average daily consumption profile for a day in the month of January. These graphs demonstrate visually how the synthetic data captures the aggregate time-of-day patterns in the empirical data. Figure 4 shows the time-series of an entire month of synthetic data. It reveals how the mixture model varies the day-to-day aggregate consumption, and smooths the transition from one month to another. Figure 5 shows that the conditional probability distribution of consumption for two months (January and February) for the empirical (real) and synthetic data are similar. Figure 6 provides a quantitative comparison of the empirical and synthetic datasets. This graph shows summary statistics (mean, median, standard deviation, minimum, maximum) of the empirical and synthetic data for four different months (September - December). This graphs shows that statistically these datasets are quite similar. The biggest difference is in the maximum value, which is always slightly higher in the synthetic data than in the real data; this is because in the TPM, Laplace smoothing introduces a non-zero probability of transition between any two states. This can be tuned depending on how closely a user needs the synthetic data to adhere to the empirical data. Figure 7 shows that the average auto-correlation for empirical and synthetic consumption time series for lags up to 24 hours is comparable. Figure 7(c) shows the auto-correlation of naively generated synthetic data, where the data is sampled independently from the consumption distribution of each hour. Although the summary statistics of this data matched well with the empirical data, as expected, the auto-correlation is not well captured.

Based on these results, we believe that our synthetic data generator provides reasonably realistic sensor data, and therefore enables us to elicit realistic behavior in experimental evaluations of any Big Data analytics platforms that we wish to examine for Internet of Things use cases.



(a) (b)

Figure 3: Consumption data for one day in January: (a) empirical; (b) synthetic.

2.3 Benchmark Queries

When embarking on this study, we were not aware of any common set of analyses (i.e., queries) that utility providers plan to run on a large resource consumption dataset. Karnouskos *et al.* state that billing is the current "killer app" [15]; this is also the analysis performed in previous smart metering benchmark studies [2, 13, 21]. Thus, this is one analysis that we consider. However, we wanted to consider additional analyses that might be of interest to utilities that would also stress the system under study in different ways.

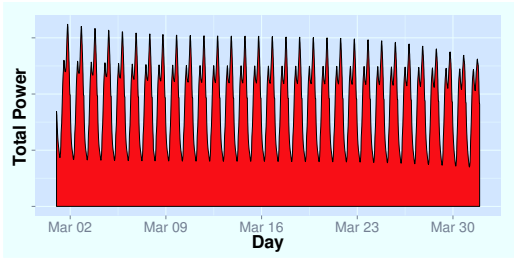


Figure 4: Synthetic consumption data for one month (March).

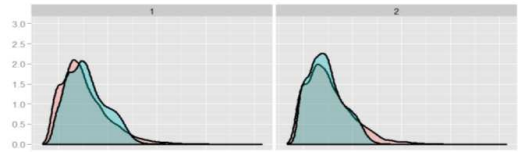


Figure 5: Comparison between power consumption distribution of synthetic and empirical data for January and February.

As shown in Table 1 we designed six distinct analyses: (1) Total readings: counts the total number of readings (i.e., rows) for the given time period. (2) Total consumption: sums the resource consumption for the given time period. (3) Peak consumption: create a sorted list of the aggregate consumption in each ten minute interval[6] in the given time period. (4) Top consumers: create a list of the distinct consumers, sorted by their total (monthly) consumption. (5) Consumption time-series: calculate the time-series of aggregate consumption per ten minute interval in the given time period. (6) Time of Usage Billing: calculate the monthly bill for each consumer based on the time of usage.

Analysis (1) is used primarily as a sanity check, to verify whether the proper amount of sensor data is being stored for the given time period. Analysis (2) determines the aggregate electricity consumption over an extended period of time (e.g., one month). Analysis (3) determines the aggregate electricity consumption in each ten minute interval over a longer duration of time (e.g., one month). It then sorts the results based on the total consumption per interval (highest to lowest). Analysis (4) looks at the aggregate consumption per meter rather than per time interval. It sorts the results based on the total consumption per meter (highest consumption to least consumption). Analysis (5) is similar to Analysis (3), except the results are kept in chronological order. This type of analysis could be used to visualize the aggregate consumption of electricity over time. Lastly, Analysis (6) calculates four "bill determinants" per consumer: the cost for electricity used during off-peak times, the cost for electricity used during peak times, the cost for electricity used during "shoulder" times,[7] and the total cost for the electricity consumed by the consumer (i.e., the sum of the first three bill determinants).

[6]Our study assumes the utility's smart meters will record a consumption value every ten minutes. Alternative interval lengths could be used in IoTAbench.

[7]"Shoulder" times refer to the transition period from off-peak to peak usage, as well as the transition period from peak to off-peak usage.

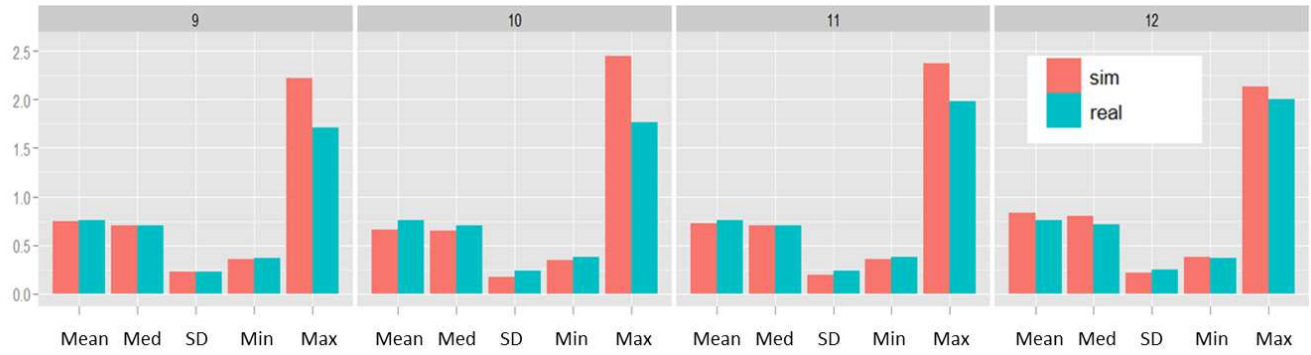
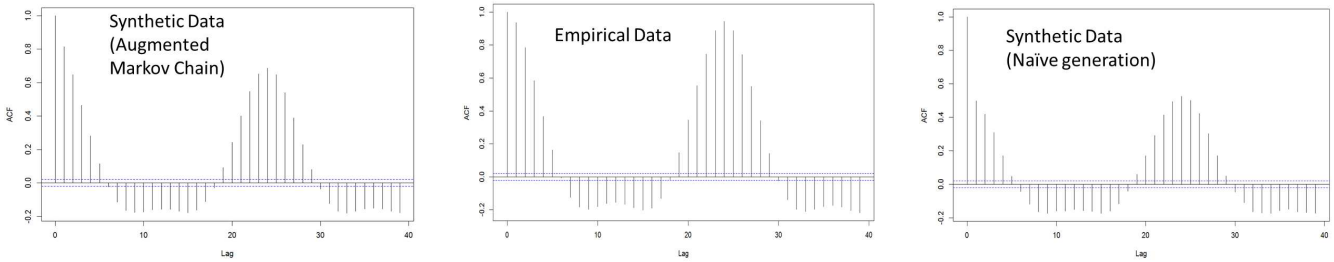


Figure 6: Comparison between summary statistics of synthetic (sim) and empirical (real) data for September through December.



(a) (b) (c)

Figure 7: Comparison between time-series auto-correlation with lags up to 24 hours; (a) IoTAbench data; (b) empirical data; (c) naively generated data.

Obviously, there are virtually an unlimited number of types of information that could be extracted from the dataset; our objective is not to demonstrate all possible queries that could be performed on the dataset. Our six queries are valid examples of information that would be relevant from a business perspective within a utility. These analyses also exercise the system under study in meaningful ways from a performance evaluation perspective. Table 1 compares the composition of the SQL queries used to perform each of the six analyses in the benchmark. In this particular study, three columns are used to store each smart meter reading: ts_key stores a timestamp value (the time the reading was taken), meter_key stores the unique identifier of the meter that prepared the reading, PowerWatts stores the consumption value for the given interval and meter. For example, the Total Readings query ("SELECT COUNT (*)") could use any of the three data columns or other means (e.g., metadata kept on the stored data) to determine the number of readings that are stored in the main table. The decision of how the query is executed is left up to the query optimizer. This query uses a WHERE clause to restrict the analysis to a specific time period; i.e., the month being analyzed. Thus, the WHERE clause makes use of the ts_key column. The Total Readings query does not involve an ORDER BY clause. Overall, the six queries use different numbers and combinations of columns in computing their output. The complexity of the queries also varies quite substantially. Table 1 helps highlight how these queries stress the system under study in different ways, in addition to providing information that is relevant to the business.

2.4 Test Harness

For Big Data studies it is particularly important that all aspects of an experiment be conducted in a systematic fashion. This aids in ensuring that the results are complete, repeatable and completed in as timely a manner as possible. We used a wrapper script around each of the tools we used (e.g., synthetic data generator, SQL client, etc.). We then created a single control script to manage the entire experiment. This control script called the wrapper scripts to invoke the specific tools as they were needed. Our control and wrapper scripts were written in Bash; we used Perl scripts to parse the experimental output. Alternative scripting languages could have provided the same functionality, we used Bash and Perl due to our familiarity with them.

To simplify the management of an experiment, as well as to make it simpler to install and use IoTAbench on a new testbed, we use a single configuration file to store all of the variables. The user edits the configuration file to set up an experiment. IoTAbench then uses the configuration file and a set of template scripts to create the scripts to be used for a specific experiment on the selected testbed.

Based on feedback from utility customers, the smart metering use case assumes batch uploads and analysis of the data. Extending IoTAbench to load and analyze streaming sensor data is ongoing as part of a separate use case.

Table 1: Comparison of benchmark queries.

Query \ Column	SELECT			WHERE			ORDER BY		
	ts_key	meter_key	powerWatts	ts_key	meter_key	powerWatts	ts_key	meter_key	powerWatts
Total Readings		any		√					
Total Consumption			√	√					
Peak Consumption	√		√	√					√
Consumption Timeseries	√		√	√			√		
Top Consumers		√	√	√	√				√
Time of Usage Billing		√	√	√	√			√	

3. EXPERIMENTAL DESIGN

3.1 Overview

The experiments described in this paper represent a single case study (smart meters) to demonstrate how IoTAbench can be used to experimentally evaluate a Big Data analytics platform.

3.2 System Under Study

Our experiments were run on a cluster of eight HP ProLiant DL380p gen8 servers. Each of these 2U servers was configured as follows: two Intel Xeon E5-2670 CPUs (each with eight 2.60 GHz cores, 20 MB cache, 8.00 GT/s QPI), 128 GB RAM, two 300 GB SAS 10K RPM drives, twenty-two 900 GB SAS 10K RPM drives, and two dual-port 10 Gb Ethernet NICs. Red Hat Enterprise Linux Server release 6.4 (Santiago) was the Operating System used on each of these servers. The two 300 GB drives were configured as RAID1, and are used for the Operating System and the database catalog directory. The twenty-two 900 GB drives were configured as RAID10 and used for the database data directory. These servers and the HP 5900AF-48XG 10 Gb network switch connecting them were used exclusively by the system under study during the experiments. On this system we installed the HP Vertica Analytics platform version 7.0.0. The experiments were controlled by a ninth DL380p server.

3.3 Design and Methodology

Table 2 lists the key characteristics of the synthetic dataset used in our study. We assume a fictitious electric utility has 40 million customers (each with their own smart meter). The utility wishes to record the consumption of each customer every 10 minutes. This corresponds to 144 readings per day per customer. The number of customers was selected such that our fictitious utility would be as large as any of the utilities in Europe. The reading interval was chosen based on the shortest interval that we heard a utility ask for (other values we heard utilities ask for were 15, 30 or 60 minutes). We store the consumption value as an integer value, rounded to the nearest Watt (if a utility desired greater precision, that could be handled in a similar manner). This is a general "best practice" for storing large volumes of numeric data. Lastly, in practice it is a common occurrence that some of the meter data is lost before it reaches the analysis platform. Smart meter vendor Itron reports 1% as a common value for how much data is typically missing [13]. Thus, we use that value as a parameter during the generation of the synthetic data.

There are several existing benchmark studies involving the collection and analysis of smart meter data [2, 13, 21]. These studies look at a month in the life of a utility scenarios, as business processes like billing often occur at this

Table 2: Synthetic dataset characteristics.

Characteristic	Setting
Number of customers	40 million
Reading interval	10 minutes
Meter precision	nearest Watt
Missing readings	1%

frequency. For this reason, our smart metering benchmark in IoTAbench uses a monthly scenario for a utility. However, previous studies look at only a single month in the life of a utility; this offers no insights into how the platform will behave over time, as the scale of data that it stores increases. Given the investments utilities must make in the metering infrastructure to collect the fine-grained consumption data, they will want to retain and use data for more than the month it was collected in. To the best of our knowledge, we are the first to explore this issue for Big Data platforms.

Based on the characteristics listed in Table 2, our dataset consists of 5.76 billion readings per day. For a month with 31 days, this is 178.6 billion readings per month. 1% of these are "missing"; in our case, the synthetic data generator never writes them to disk. The platform under study therefore needs to repair the dataset to return to the expected number of readings for the month.

Our experimental methodology is as follows. For each month of data we:

- generate the synthetic dataset for that month
- load the "raw" data into a staging area
- repair the "raw" data and store the repaired data in the main table
- delete the "raw" data and empty the staging area
- analyze the repaired data

This process is continued until the platform runs out of storage space for new data. Once this process finishes, we perform a final step, a"re-analysis" of each month of data. This final step is done to determine if the platform has made any tradeoffs in how it manages the data over time. For example, is older data penalized in an attempt to keep newer data quick to access?

An important feature of the specific platform we tested (HP Vertica Analytics Platform) is the ability to optimize the layout of the data on disk. This requires having a sample of the data for the designer and optionally one or more sample queries to use to calculate the best design to use. In our experiments we ran the designer after the first month of data was loaded, and optimized for the Consumption Timeseries query. With HP Vertica it then automatically stores

all subsequent data that is loaded into the optimized layout; i.e., the design only needs to be determined once. In our experiment with one month of data, this optimization step took 5.25 hours. This is a relatively small overhead given the benefits it provides. In addition, the same design could have been obtained using less data, requiring less time. However, since we were using one month of data in all of our other steps, we did for the optimization step as well.

A challenge to address in the development of an industry standard IoT benchmark is how to accommodate "special features" like HP Vertica's Data Designer. While the generation of data and the analysis queries (written in SQL) in IoTAbench should be portable across SQL-enabled Big Data platforms, the queries to load data, repair missing data or optimize the data layout will not. However, if one only considers industry-standard features, the benchmark may not adequately capture the capabilities of the platform. A reasonable tradeoff might be to distinguish between "standard" and "non-standard" performance results, much like publicly-traded companies may report GAAP (Generally Accepted Accounting Principles) and non-GAAP financial results.

4. EXPERIMENTAL RESULTS

4.1 Generator Performance

Since our experiments involve the generation of massive amounts of data, the performance of our synthetic data generator is important. Our implementation takes advantage of the fact that the process is embarrassingly parallel and that our testbed uses a distributed set of multi-core servers.

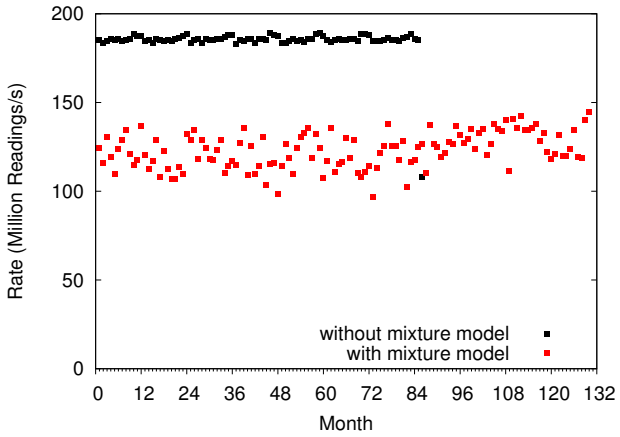


Figure 8: Generator performance.

Figure 8 shows the performance of our generator. Our initial version did not include the mixture model to smooth transitions between months. In version 1 we were able to consistently generate 185 million meter readings/second (≈ 6 GB/s of data). At this rate we could generate a month of data for 40 million meters in about 16 minutes. The addition of the mixture model requires the use of considerably more random numbers, which lowered the average generation rate to 120 million meter readings/second, and made the performance much more variable (the random number generation is a bottleneck). Since this still allowed us to generate a month of data in about 25 minutes, we did not attempt to alleviate the bottleneck, although that could be a topic of future work.

4.2 Load and Repair Performance

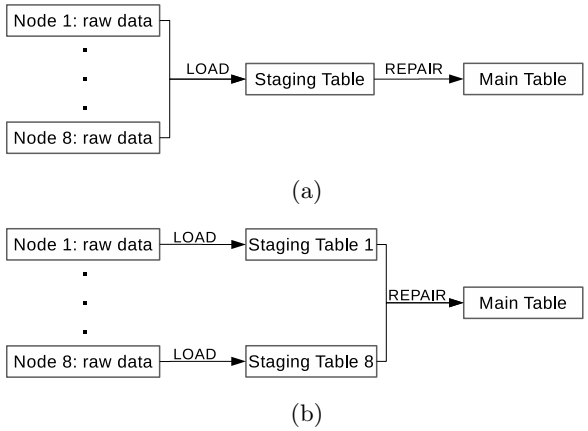


Figure 9: Load implementations considered in our study: (a) scale-up; (b) scale-out.

Figure 9 shows two different load options that we considered. The *scale-up* implementation loads the files on each node into a single staging table. The repair step then reads the raw meter data from the staging table, repairs it, and inserts it in the main table. The *scale-out* implementation involves each cluster node loading the files into a distinct staging table, i.e., one that no other node is directly loading raw data into. The repair step then merges the data from the eight staging tables, repairs it, and inserts the aggregated data into the main table. Once the repaired data is loaded into the main table, the staging area is emptied.

In practice, data is often imperfect. Thus, a Big Data analytics platform must be capable of identifying and repairing data quality issues with the raw data. Since a smart meter use case involves time series data, we perform this step by interpolating any missing values. Defining a more extensive repair methodology is left for future work.

Figure 10(a) shows the rate at which data was loaded into the staging tables. In experiment 1, the scale-out implementation described in Figure 9(a) was used. In experiment 2, each cluster node loads data into a distinct staging table, as explained in Figure 9(b); the scale-out load approach. Figure 10(a) reveals that the scale-out approach offers a significant performance advantage, achieving an average load rate of 39.4 million readings/second compared to an average of 11.7 million readings/second for the scale-up approach, a difference of 3.37x. Figure 10(a) reveals that the load performance of either implementation is quite consistent over time, even as the cluster has used nearly all of the available storage space.[8]There is more variability in the load results when using the scale-out load approach, although this ap-

[8]The experiments concluded in the 87th month for the first experiment and in the 131st month for the second experiment. The different durations were due to different data designs affecting how quickly storage space was consumed. We verified that the bottleneck in the scale-up experiment was from loading to the shared staging area, and not from using a different data design.

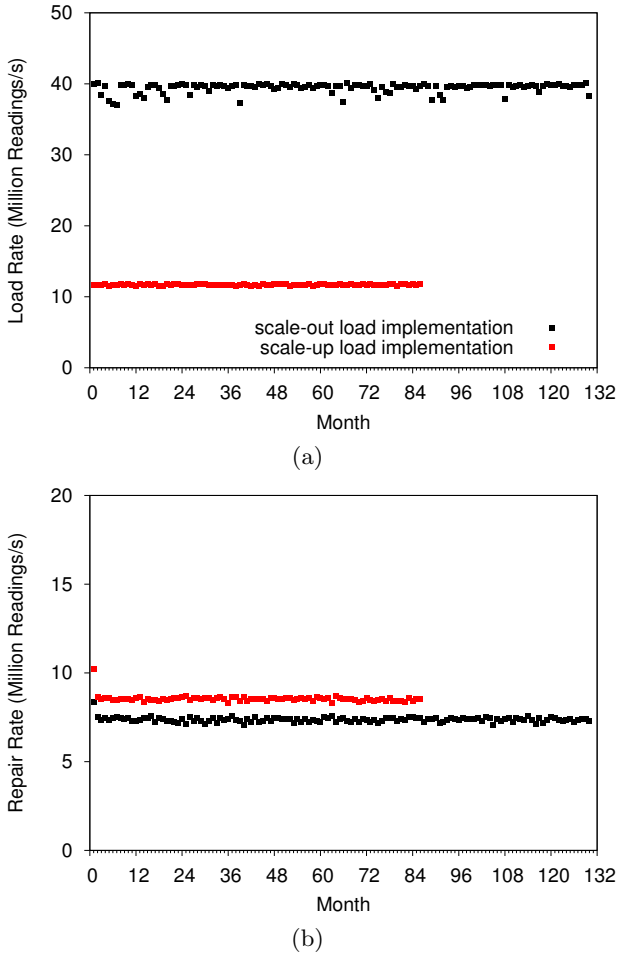


Figure 10: (a) Load Performance; (b) Repair Performance

proach still has much better overall performance than the scale-up approach.

The repair rate for the two different load approaches is shown in Figure 10(b). For both experiments the repair rate for the first month was higher than for the remaining months; this is due to the data layout being optimized after the first month of data is stored in the main table. Excluding the repair time for the first month, the median repair rate in the first experiment was 8.5 million readings/second, while in the second experiment the median repair rate was 7.4 million readings/second, about 15% slower. There are two potential causes of the slowdown; a) the more efficient encoding method used on the meter_key column in the second experiment requiring more time to compress the data, and b) the need to merge the raw data from eight staging tables before repairing it in the second experiment. We have not attempted to quantify the extent that each of these contributed to the 15% overhead, as this is a relatively small overhead compared to the increased data retention that was achieved and the increased load rate the scale-out approach offers. Overall, we were able to load 22.8 trillion readings and 726.9 TB of data in the second experiment, which Vertica compressed and stored in 65.4 TB of disk space. This is a compression factor of 11.12x (uncompressed size/compressed size). This is substantially more

Table 3: Properties of datasets used in each experiment.

Property	Experiment 1	Experiment 2
Total Readings	15.1 trillion	22.8 trillion
Duration	7 years 2 months	10 years 10 months
Total Size	478 TB	727 TB

data than we were able to load in the first experiment (15.1 trillion readings and 478 TB of data). The primary cause is the improved compression of the meter_key column.

Table 3 summarizes the characteristics of the datasets we used in each experiment. It is important to note that the experiments were run with application-level data redundancy turned on. This means that the system under study retained two copies of the data so that it could withstand the failure of one cluster node. Thus, while our system under study in the second experiment stored 22.8 trillion *distinct* readings, it actually stored 45.6 trillion *total* readings (two copies of each distinct reading, intelligently placed around the cluster to provide the ability to keep the cluster operational in the event of a node failure).[9]

4.3 Analysis Performance

After loading and repairing each month of data the next step in the benchmark is to run a set of analyses on it. IoTAbench cycles through each analysis, then repeats the cycle twice and records the median value. After completing the analysis of the final month of data that the system is able to load, the analyses are re-run on each month of data. This re-analysis step is included in the benchmark to assess whether the Big Data Analytics Platform under study has made any trade-offs that may affect query performance, particularly on older portions of the dataset. The re-analysis step also performs the entire set of benchmark queries, then repeats twice and reports the median query time.

In the remainder of this section we focus on the performance results for Experiment 2, which stored 130 months of data in total.

4.3.1 *Total Readings Performance*

Figure 11 shows the performance of the Total Readings analysis when run on our system under study. Note that the y-axis is in logarithmic scale, to enable a better comparison of the range of query times. Prior to optimizing the data design, the Total Readings analysis took 23.996 seconds to count 178.6 billion readings. After the re-design, the analysis of the same data (Month 1) took 20.239 seconds. Over the entire 130 months of data (black squares), the median time for this analysis was 19.302 seconds, while the average was 19.165 seconds. For this specific query, the new data layout had only a minimal positive effect on the query times.

There is more than one way to perform most analyses. In this case, "SELECT COUNT(*)" was our initial implementation to determine the number of readings. An alternative method is to use a specific column to use to determine the same result; e.g., "SELECT COUNT(ts_key)". One motivation for considering these different implementations of this simple analysis is that enables one to quantify the implications on query performance of the different encoding

[9]In addition to application-level redundancy, our cluster has storage-level RAID 10 redundancy to retain the data in the event of a disk failure.

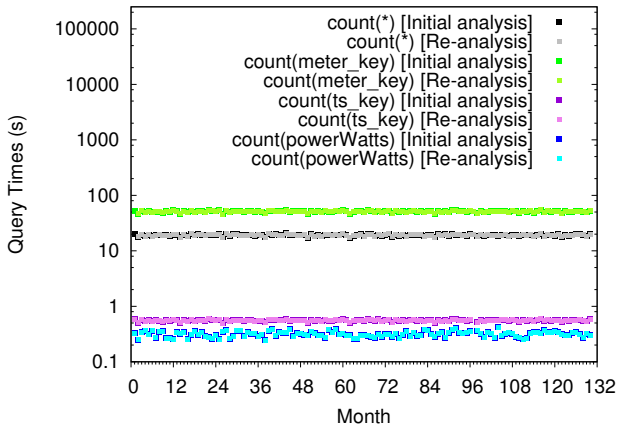


Figure 11: Performance of Total Readings query.

methods used for the different columns. For example, performing this query on the powerWatts column (dark blue squares) took a median time of 310 ms to complete each month. This query took slightly longer on the ts_key column (dark purple squares), averaging 555 ms (558 ms median). The query took the longest on the meter_key column (dark green squares), with a median time of 51.301 seconds. This illustrates the tradeoff for achieving better compression of the meter_key column and the dataset overall; it takes a lot longer to access the meter_key data. As a result, any queries that use this column will encounter this overhead.

Figure 11 also shows the performance results of the TotalReadings query and its variants during the re-analysis stage. Figure 11 shows that the TotalReadings analyses are just as fast in the re-analysis stage. This demonstrates that the system under study performs in a consistent manner as the data sizes grow; i.e., it does not make any tradeoffs to keep queries fast on the most recently added data that might penalize queries involving older data.

4.3.2 Total Consumption

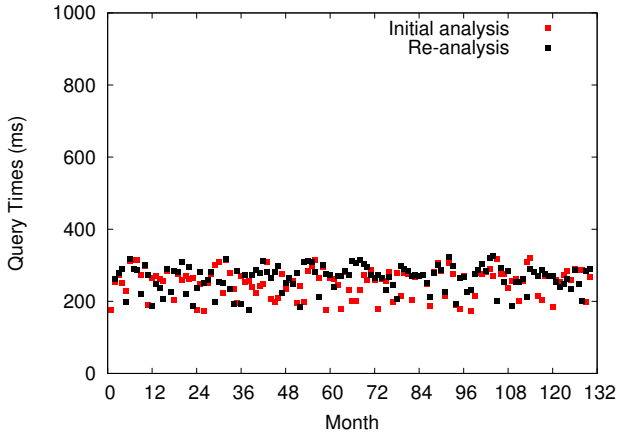


Figure 12: Performance of Total Consumption query.

Figure 12 shows the performance results for the Total Consumption query. Initially, the Total Consumption analysis for the first month of data took 49.813 seconds. After optimizing the data layout, the analysis on the same data took 0.176 seconds. Over the 130 months of data, the average time for this query was 0.249 seconds. In the re-analysis step, the results are quite similar to the performance when the analysis was initially run. The average time for this query was 0.264 seconds. Note that on a linear scale the results appear to have quite a bit of variability in them. This is due at least in part to distributing the query across eight different nodes. Nevertheless, the query times are very low in all cases.

4.3.3 Peak Consumption

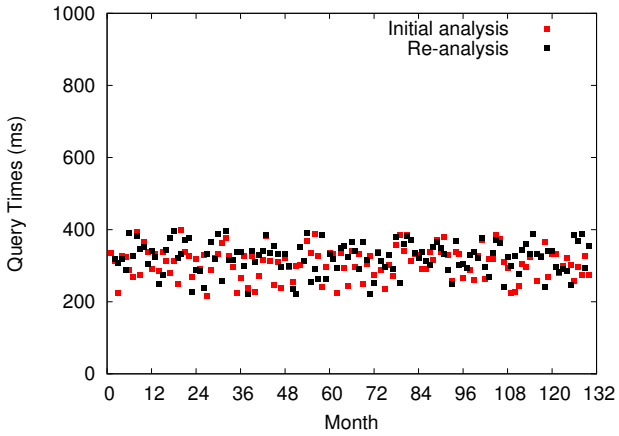


Figure 13: Performance of Peak Consumption query.

Figure 13 shows the results for the Peak Consumption query. The peak consumption analysis took 92.862 seconds prior to the data layout optimization. This was reduced to 0.334 seconds afterward. Across all 130 months, the average was 0.307 seconds. The query times for the re-analysis step are quite consistent with the initial results, with an average time of 0.322 seconds to run the Peak Consumption query.

As shown in Table 1, the Peak Consumption analysis is similar to the Consumption Timeseries query, the only difference being how the results are ordered. After running the experiments we found that there was minimal difference in the performance. Thus, the results in Figure 13 are quite similar to those for the Consumption Timeseries analysis (Figure 14).

4.3.4 Consumption Timeseries Performance

Figure 14 shows the performance of the Consumption Timeseries analysis. The initial query time was 92.639 seconds. After optimizing the data layout, the query time dropped substantially. The median value over 130 months was 0.355 seconds. The low response time and variability both suggest that a user could potentially interact with the data, which could enable real-time exploration of a large sensor dataset. This is a significant result. A month of raw data in our utility use case is nearly 6 TB in size; being able to create an aggregate timeseries from it (e.g., Figure 4) in only a few hundred milliseconds suggests that business analysts could interact with the data, such as visually examining regions of consumption activity that are interesting from a business perspective (e.g., zooming in and out of time periods with

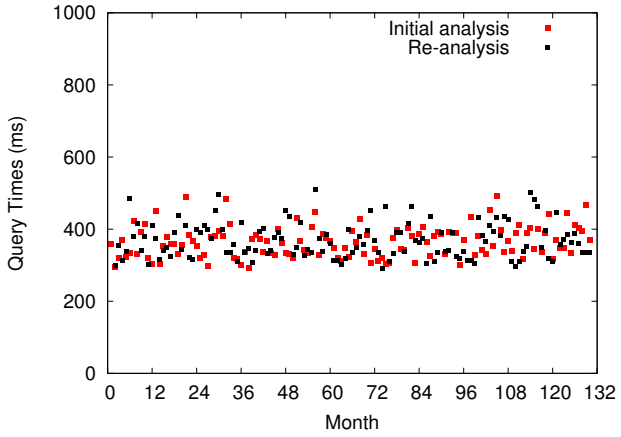


Figure 14: Performance of Consumption Timeseries query.

abnormal consumption patterns, drilling down into who is contributing to spikes in consumption, etc.).

The re-analysis query times are also shown in Figure 14. Once again, the re-analysis query times are quite similar to the initial query times. The median query time was 0.361 seconds. The variability in the re-analysis results is quite low. These results indicate that a business analyst will be able to interact with historical data about their business in the same way that they can examine current data. This is helpful for identifying emerging trends in their business.

4.3.5 *Top Consumers Performance*

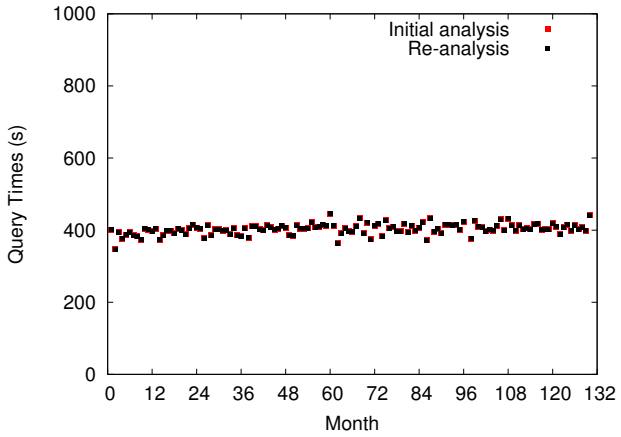


Figure 15: Performance of Top Consumers query.

Figure 15 shows performance results for the Top Consumers analysis. The median query time during the initial analysis was 401.997 seconds. The average re-analysis query time was 408.968 seconds, which is slightly slower (1.7%) than in the initial case, but still relatively the same as when the data was first added to the database.

4.3.6 *Time-of-Usage Billing Performance*

Our final analysis is Time of Usage Billing. The results are presented in Figure 16. As with Top Consumers, we expected to sacrifice some performance on this query (since it involves the meter_key column) in order to be able to store a longer duration of data.

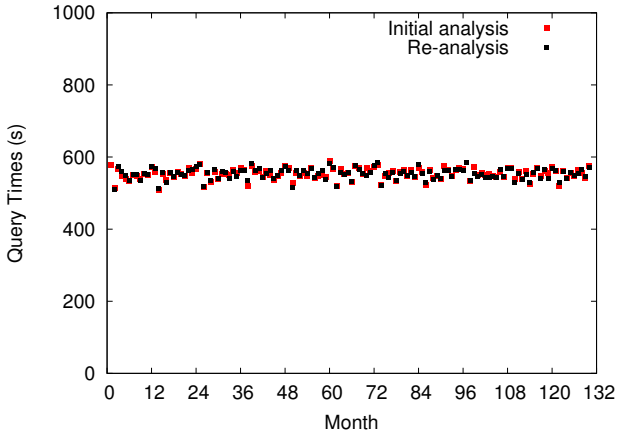


Figure 16: Performance of Time-of-Usage billing query.

The median value over the 130 months was 554.395 seconds. The re-analysis results (also shown in Figure 16) reveal that the query times on older data complete about as quickly as when the queried data was initially added to the database. The median query time for the re-analysis was 555.631 seconds, which is essentially identical to the initial case. This demonstrates that with the system under study, an organization could work with their historical data just like it is new data.

4.3.7 *Analyzing the Entire Dataset*

Table 4: Benchmark query times over entire dataset.

Query	Time (s)
Total Readings	10.859
Total Readings (ts_key)	10.131
Total Readings (meter_key)	6,673.220
Total Readings (powerWatts)	11.549
Total Consumption	10.636
Peak Consumption	149.034
Consumption Timeseries	155.412
Top Consumers	51,135.932
Time of Usage Billing	54,695.257

As a final exercise, we used IoTAbench to analyze the entire dataset. We modified the benchmark queries so that each would consider the entire dataset (i.e., all 22.8 trillion readings). The query results may not be particularly useful from a business perspective, but the performance results are interesting for understanding whether a "Big Data platform" scales when analyzing a larger dataset. The main modification required to each of the queries was to remove the WHERE clause that filters the data to be considered.

Table 4 provides the results of this exercise. The results can be grouped into three categories. First, two analyses (Total Readings (meter_key) and Top Consumers) had performance that was reasonably close to 130 times the average time they required on a single month of data. These queries involved decompressing the heavily compressed meter_key column, making the query time quite predictable.

The second category includes those where the actual time is much lower than 130 times their average monthly time. This includes Total Readings, Total Readings (ts_key), Total Readings(powerWatts), Total Consumption, and Time of Usage billing. Eliminating the WHERE clause is responsible for at least part of the speedup. For the Time of Usage billing query, the results only need to be sorted once, rather than once per month. The third category are the analyses that take longer than 130 times their monthly average.In this category are Peak Consumption and Consumption Timeseries. These two queries need to keep state on an increasingly larger number of intervals. If a utility wished to search larger intervals (e.g., hours or days) rather than the default ten minute interval when analyzing the entire dataset, then the performance of this sort of analyses may improve. Alternatively, for these queries one could improve the time by issuing each of the monthly queries in sequence and concatenating the results together.

5. RELATED WORK

The most closely related work to ours are white papers from IBM, Microsoft, and Oracle, examining how their products perform for storing, repairing and analyzing smart meter data [2, 13, 21]. Our study uses a dataset that is 2-3 orders of magnitude larger than these studies, uses a larger set of queries, and explores performance and scalability of the system under study in much greater depth.

The "Internet of Things" is a popular topic in research and business circles alike. However, much of the discussion is focused on device features and connectivity issues rather than on managing and analyzing the large datasets that will be collected. The most related work we have come across in this space is by Ding *et al.*, who propose a "statistical database cluster mechanism for Big Data analysis in the Internet of Things" [7]. They provide a small-scale performance evaluation of their framework. Ma *et al.* consider an indexing mechanism for "massive" IoT data in a cloud environment [19]. Their performance evaluation involves four orders of magnitude fewer records that our study.

"Big Data" is a popular topic that overlaps the Internet of Things but spans other areas as well. An issue in the "Big Data" domain is that there are different interpretations of what constitutes "Big". Descriptions of "Big Data" systems in production environments typically mention data sizes in the hundreds of TB to hundreds of PB [27, 4, 17] or trillions to hundreds of trillions of rows [12]. "Big Data" research studies on the other hand tend to work with much smaller datasets, ranging from hundreds of GB [6, 18, 16, 24] to a few TBs [29, 28, 23, 1]. The largest dataset we have seen used in an existing "Big Data" study is 16 TB [9]. We attempt to bridge the gap between research and production in our paper, working with a dataset that is three-quarters of a PB in size on a modest-sized cluster.

Benchmarking is another related domain. There are a number of groups like SPEC[10] and TPC[11] that create industry standard benchmarks. At this time we are not aware of any industry standard IoT benchmarks. BigBench is a recent effort to create an industry standard benchmark for the "Big Data" domain [11]. That work considers a broader scope than we are; however, there may be an opportunity to collaborate with them. Other recent works include a social graph benchmark [3] and benchmarks for cloud services [5, 26, 22]. While these could potentially be used to benchmark "Big Data" applications, they are not specifically focused on IoT applications.

There is a long history of research on predicting electricity load curves and the factors that affect them. For example, Gellings and Taylor developed a simulation model of a utility's load shape in 1981 [10]. 70 years ago, Dryar described a method to predict the effect of weather on a utility's load [8]. The key difference between such works and ours is that we are generating the loads for individual households that once aggregated provide the desired utility load.

6. CONCLUSIONS

This paper introduced a benchmark called IoTAbench for evaluating Big Data analytics platforms for the Internet of Things. To generate a large data set with realistic properties, we used our Markov chain-based synthetic data generator. To demonstrate the potential for IoTAbench to benchmark Big Data IoT applications, we considered a smart metering use case, and evaluated the HP Vertica 7 Analytics platform for a scenario involving an electric utility with 40 million meters.

To our knowledge, this is one of the largest Big Data research studies to date, involving 22.8 trillion distinct readings and 727 TB of raw data. In our opinion, this work helps bridge the divide between "Big Data" research and practice. In other words, we provide practitioners with insights into a platform managing sensor data at production scale, and simultaneously demonstrate to other researchers a methodology to follow to conduct their research at production scale. It is important to note that this work was done on an eight-node cluster, not a cluster of hundreds or thousands of servers as is often mentioned in discussions of production environments.

We would like to contribute our tools and experience towards the creation of an industry standard IoT benchmark suite. We plan to identify additional IoT applications and sensor data types to include as part of that suite.

Acknowledgments

The author would like to thank Priya Arun, Cullen Bash, Sue Charles, Carlos Felix, John Gearen, Meichun Hsu, Uri Kogan, Harumi Kuno, Oliver Moreno, Brad Morrey, Alison Reynolds, Stewart Robbins, Mark Slavin, and Min Xiao for their constructive feedback and assistance.

[10] `http://www.spec.org`
[11] `http://www.tpc.org`

7. REFERENCES

[1] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz. Hadoop-GIS: A high performance spatial data warehousing system over MapReduce. In *Proceedings of the VLDB Endowment*. VLDB Endowment, August 2013.

[2] AMTSybex and IBM. Sink or swim with smart meter data management. September 2011.

[3] T. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan. LinkBench: a database benchmark based on the Facebook social graph. In *SIGMOD 2013*, pages 1185–1196. ACM, June 2013.

[4] D. Borthakur. Petabyte scale databases and storage systems at Facebook. In *SIGMOD 2013*. ACM, June 2013.

[5] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *SoCC'10*, June 2010.

[6] E. Dede, Madhusudhan, D. Gunter, R. Canon, and L. Ramakrishnan. Performance evaluation of a MongoDB and Hadoop platform for scientific data analysis. In *ScienceCloud 2013*, June 2013.

[7] Z. Ding, X. Gao, J. Xu, and H. Wu. IOT-StatisticDB: A general statistical database cluster mechanism for big data analysis and the Internet of Things. In *IEEE Internet of Things*. IEEE, August 2013.

[8] H. Dryar. The effect of weather on the system load. *AIEE Transactions*, 63(12):1006–1013, December 1944.

[9] A. Floratou, N. Teletia, D. Dewitt, J. Patel, and D. Zhang. Can the elephants handle the NoSQL onslaught? In *Proceedings of the VLDB Endowment*, pages 1712–1723. VLDB Endowment, August 2012.

[10] C. Gellings and R. Taylor. Electric load curve synthesis - a computer simulation of an electric utility load shape. *IEEE Transactions on Power Apparatus and Systems*, PAS-100(1):60–65, January 1981.

[11] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H. Jacobsen. BigBench: towards an industry standard benchmark for big data analytics. In *SIGMOD 2013*, pages 1197–1208. ACM, June 2013.

[12] A. Hall, O. Bachmann, R. Bussow, S. Ganceanu, and M. Nunkesser. Processing a trillion cells per mouse click. In *Proceedings of the VLDB Endowment*, pages 1436–1446. VLDB Endowment, August 2012.

[13] Itron and Microsoft. Benchmark testing results: Unparalleled scalability of Itron Enterprise Edition on SQL Server. May 2011.

[14] D. Jones and M. Lorenz. An application of a Markov chain noise model to wind generator simulation. *Mathematics and Computers in Simulation*, 28:391–402, 1986.

[15] S. Karnouskos, P. G. da Silva, and D. Illic. Assessment of high-performance smart metering for the web service enabled smart grid. In *ICPE 2011*, March 2011.

[16] M. Kaufmann, A. Manjili, R. Vagenas, P. Fischer, D. Kossmann, F. Farber, and N. May. Timeline index: A unified data structure for processing queries on temporal data in SAP HANA. In *SIGMOD 2013*. ACM, June 2013.

[17] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear. The Vertica analytic database: C-store 7 years later. In *Proceedings of the VLDB Endowment*, pages 1790–1801. VLDB Endowment, August 2012.

[18] P. Larson, C. Clinciu, C. Fraser, E. Hanson, M. Mokhtar, R. Rusanu, and M. Saubhasik. Enhancements to SQL server column stores. In *SIGMOD 2013*. ACM, June 2013.

[19] Y. Ma, J. Rao, W. Hu, X. Meng, X. Han, Y. Zhang, Y. Chai, and C. Liu. An efficient index for massive IOT data in cloud environment. In *CIKM'12*, October 2012.

[20] C. MacGillvary, V. Turner, and D. Lund. Worldwide Internet of Things (IoT) 2013-2020 forecast: Billions of things, trillions of dollars. *IDC*, pages 1–22, October 2013.

[21] Oracle. Meter-to-cash performance using Oracle Utilities applications on Oracle Exadata and Oracle Exalogic. January 2012.

[22] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. Lopez, G. Gibson, A. Fuchs, and B. Rinaldi. YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In *SoCC'11*, October 2011.

[23] A. Pavlo, E. Paulson, A. Rasin, D. Abadi, D. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *SIGMOD 2009*. ACM, June 2009.

[24] V. Raman, G. Attaluri, R. Barber, and N. Chainani. DB2 with BLU acceleration: so much more than just a column store. In *Proceedings of the VLDB Endowment*. VLDB Endowment, August 2013.

[25] D. Shamshad, M. Bawadi, W. Hussin, T. Majid, and S. Samusi. First and second order Markov chain models for synthetic generation of wind speed time series. *Energy*, 30:693–708, 2005.

[26] Y. Shi, X. Meng, J. Zhao, X. Hu, B. Liu, and H. Wang. Benchmarking cloud-based data management systems. In *CloudDB'10*, October 2010.

[27] J. Shute, R. Vingralek, B. Samwel, B. Handy, C. Whipkey, E. Rollins, M. Oancea, K. Littlefield, D. Menestrina, S. Eliner, J. Cieslewicz, I. Rae, T. Stancescu, and H. Apte. F1: A distributed SQL database that scales. In *Proceedings of the VLDB Endowment*. VLDB Endowment, August 2013.

[28] P. Wong, Z. He, and E. Lo. Parallel analytics as a service. In *SIGMOD 2013*, pages 25–36. ACM, June 2013.

[29] R. Xin, J. Rosen, and M. Zaharia. Shark: SQL and rich analytics at scale. In *SIGMOD 2013*. ACM, June 2013.