Analysis of TPCx-IoT: The First Industry Standard Benchmark for IoT Gateway Systems

Meikel Poess*, Raghunath Nambiar†, Karthik Kulkarni†, Chinmayi Narasimhadevara†,
Tilmann Rabl‡ and Hans-Arno Jacobsen§
*Server Technologies, Oracle, Redwood Shores, California 94065, USA
Email: meikel.poess@oracle.com
†Unified Computing Systems, Cisco, San Jose, California 95134, USA
Email: {rnambiar,kakulkar,cnarasim}@cisco.com
‡DIMA Group, TU-Berlin, Einsteinufer 17, 10587 Berlin, Germany
Email: rabl@tu-berlin.de
§Application and Middleware Systems Group, TU-Munich, Arcisstrasse 21, 80333 Munich, Germany
Email: jacobsen@in.tum.de

***Abstract*—By 2020 it is estimated that 20 billion devices will be connected to the Internet. While the initial hype around this Internet of Things (IoT) stems from consumer use cases, the number of devices and data from enterprise use cases is significant in terms of market share. With companies being challenged to choose the right digital infrastructure from different providers, there is an pressing need to objectively measure the hardware, operating system, data storage, and data management systems that can ingest, persist, and process the massive amounts of data arriving from sensors (edge devices). The Transaction Processing Performance Council (TPC) recently released the first industry standard benchmark for measuring the performance of gateway systems, TPCx-IoT. In this paper, we provide a detailed description of TPCx-IoT, mention design decisions behind key elements of this benchmark, and experimentally analyze how TPCx-IoT measures the performance of IoT gateway systems.**

***Index Terms*—performance measurement; benchmark; performance methodologies; Internet of Things; TPC;**

I. Introduction

Without doubt, the Internet of Things (IoT) has been an influential key driver of innovation, both in the consumer and the business segments of many industries. The initial hype around the IoT has stemmed from typical consumer use cases such as wearable fitness trackers, smart watches, and smart home devices. However, the number of devices and data from enterprise use cases such as smart city, patient healthcare, preventative maintenance, and smart power grid have a significant market share. Regardless of use case, IoT will continue to drive innovation of physical devices, networks, and back-end analytical infrastructure. According to a 2017 projection by Gartner [1], the total number of IoT devices will more than double between now and 2020 (8.4 Billion to 20.4 Billion). The breakdown between consumer and business devices will remain steady at 62% and 38%, respectively.

Figure 1 shows a simplified three-tier architecture of a typical IoT system with *edge devices* on the left, *gateways* in the middle, and *datacenter* on the right. Data flows with very high frequencies and very low latency requirements from a myriad of sensors into edge devices, where the, typically,

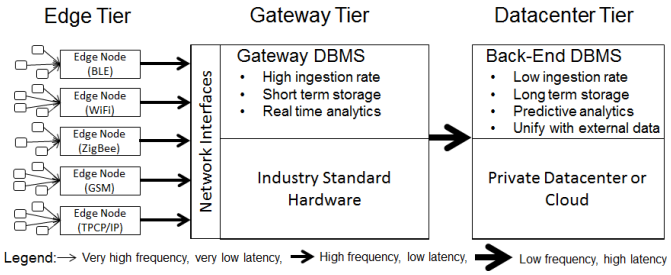

Fig. 1: Schematic overview of a typical IoT infrastructure

analog signals are converted into digital data. The frequencies at which sensors generate data depend on the sensor type. For instance, a *phasor measurement unit* (PMU), which measures electrical waves using synchrophasors [2], can generate 60-121 readings per second. Vibration sensors for monitoring machine health as part of predictive machine maintenance are capable of generating data at a much higher rate of thousands of samples per second (sps). The edge devices then send that data at high frequencies with low latency requirements by using a variety of network protocols, such as TCP/IP, Bluetooth Low Energy (BLE) [3], ZigBee [4], Wireless Internet (WiFi) [5], Global System for Mobile (GSM) [6], and MQTT [7], to gateway systems. Acting as a single point of access for these devices, gateways perform the functions of short-term persistent storage and lightweight local analytics (filter, aggregation) in their own DBMS, which often runs open-source software on industry standard servers[1]. With these capabilities, gateways serve as a crucial monitoring tool for a selected area of an operational field by providing dashboard-like functionality. The back-end system on the right side of Figure 1 ingests data from, potentially multiple gateways at low frequencies, for example, once a day, without much latency requirement. It stores data long-term and performs complex global analytics.

Analytics that run on gateways are restricted to the data

[1]Usually x86-based servers built with commodity hardware parts.

sent from the edges connected directly to a given gateway. Typical analytical queries in gateways run frequently and include data aggregation, duplicate removal, and outlier and error detection. They usually operate on a subset of the data, for example, they are restricted to a specific sensor or date range (ingested within the previous minutes). Because they are run very frequently and, potentially, concurrently with other queries, they are required to be completed in sub-second.

Analytics that run on datacenters usually span multiple gateways and consider data from previous weeks to years. They include both reporting and ad-hoc queries, which tend to be complex, long-running and based on a large subset of the data. Usually, few queries run concurrently, and they are allowed to run for an extended period.

Foreseeing a massive increase in the volume of data originating from edge devices that needs to be processed by gateway systems with high reliability and performance, the Transaction Processing Performance Council (TPC) [8] developed the TPC Express Benchmark^{TM} IoT (TPCx-IoT) [9], the first industry standard benchmark for gateway systems. Its first version was released in May 2017. Following the tradition of previous TPC benchmarks TPCx-IoT provides an objective measure of hardware, operating system, data storage, and data management systems to the industry and academia with verifiable performance, price-performance, and availability metrics.

TPCx-IoT specifically targets the gateway tier because the edge and the datacenter tiers are already covered by existing industry standard benchmarks. IoTMark, developed by Embedded Microprocessor Benchmark Consortium (EEMBC), is a suite of IoT connectivity benchmarks for testing and analyzing microcontrollers and connectivity interfaces of edge devices. The datacenter tier is covered by the TPC's own benchmarks TPC-H and TPC-DS. Covering gateway systems seemed a natural fit for the TPC because it focuses on data intensive applications. TPC discussed combining benchmarks from all three tiers into one end-to-end benchmark, but it concluded that there would be too many different devices and software solutions distributed by different vendors in such an end-to-end benchmark. This would make benchmarking very time consuming and expensive. In addition, interpretation of the benchmark results would be extremely difficult.

TPCx-IoT provides an objective measure of hardware, operating system, data storage, and data management systems to offer the industry with verifiable performance, price-performance, and availability metrics for systems that are meant to ingest and persist massive amounts of data from large a number of devices and provide real-time insights, typical in IoT gateway systems running commercially available systems, both software and hardware. The TPCx-IoT benchmark models a continuous system available 24h a day, 7 days a week. It can be used to assess a broad range of system topologies and implementation methodologies in a technically rigorous, directly comparable, and vendor-neutral manner.

The contributions of this paper can be summarized as follows.

1) We introduce TPCx-IoT.
2) We provide a detailed description of TPCx-IoT beyond what is available in its specification, including design rationales that lead to key decisions.
3) We explain how the ideas of TPCx-IoT are embedded in the TPC infrastructure that makes TPCx-IoT a robust industry standard benchmark.
4) We analyze how TPCx-IoT measures the performance of IoT gateway systems by running it against different hardware configurations.

The remainder of this paper is divided into five sections. The second section describes related work. The third section describes TPCx-IoT in detail, including its use case, execution rules, metrics, data ingestion workload, query workload and workload driver. The fourth section explains how TPCx-IoT is embedded in TPC's benchmark infrastructure including a description of its benchmark class, and its pricing and auditing requirements. In the fifth section we demonstrate how TPCx-IoT measures the performance of real systems by analyzing TPCx-IoT performance data gathered on different hardware configurations running HBase. We conclude our paper with a summary of our results.

II. Related Work

While there have been industry standard benchmarks for measuring the performance of IoT edge devices and datacenter analytic systems, TPCx-IoT is the first industry standard benchmark for measuring the performance of IoT gateway systems. Because TPCx-IoT is an express benchmark that provides a working kit, the TPC specification is limited to providing high-level information. This paper provides a detailed description of TPCx-IoT. This paper describes TPCx-IoT to an extent that is beyond what is available in the TPC specification.

The nonprofit Embedded Microprocessor Benchmark Consortium (EEMBC) has developed and is actively supporting *IoTMark* [10], a suite of micro-benchmarks for analyzing edge devices. The benchmark suite focuses on measuring the energy consumption of the three main parts of an edge node, *sensors*, *processing*, and *communication protocol*. IoTMark is based on real-world use cases and, therefore, determines the combined energy consumption of the entire edge platform (sensor interface, processor, and radio interface). However, EEMBC does not include any performance or price-performance metric in contrast to TPCx-IoT.

IoTABench [11] is an experimental benchmark toolkit. It currently implements a smart meter use case (electricity) [12], which includes the loading of synthetic data, performing simple data cleansing, and six analytical queries. The synthetic data is generated by a Markov chain data generator that was trained using real data from Irish households. The data-cleansing step requires insertion of missing data (1% dropped samples) and the six analytical queries perform *projection*, *aggregation*, *selection*, and *order by* operations. The sophisticated data generator of IoTBench generates very realistic data in parallel. IoTABench models a data ingestion rate of about

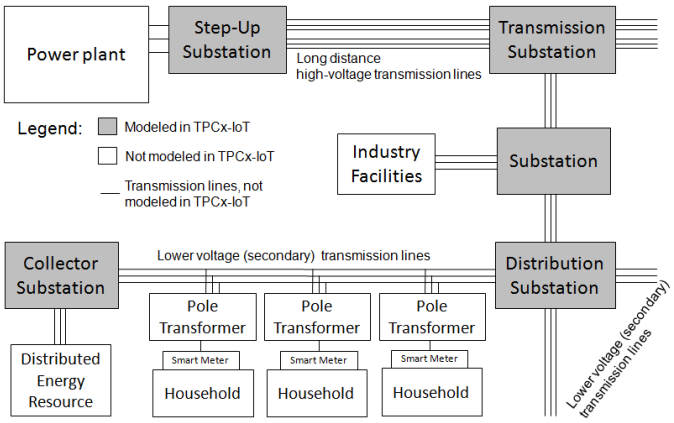

Fig. 2: High level description of TPCx-IoT's use-case

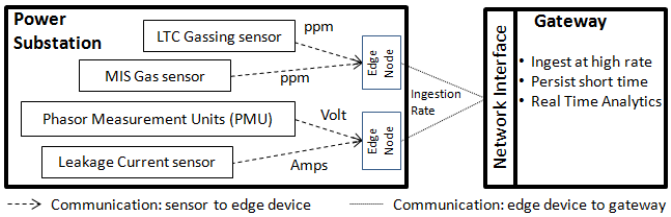

Fig. 3: Overview of a gateway architecture serving power substations

67,000 smart meter readings per second. In comparison, TPCx-IoT models millions of inserts per second. IoTABench models a single sensor type, while TPCx-IoT models 200 sensor types. IoTABench measures the data ingestion rate and query performance sequentially, while TPCx-IoT measures insert and query operations concurrently. The number of queries executed by the two benchmarks is also different. While TPCx-IoT executes five queries for every 10,000 sensor readings, IoTABenc executes a total of 12 queries (two times six different queries). With this model, IoTABench can be considered a datacenter and not a gateway benchmark. In addition, IoTABench is not an industry standard benchmark.

TPC-H [13] and TPC-DS [14], [15] are two industry standard benchmarks that are used extensively in industry and academia for measuring the performance of complex analytical systems because they are deployed in IoT datacenter back-end systems. TPC-H being the older of the two standards requires an ACID compliant database system, while TPC-DS does not, because it specifically targets big data systems.

III. Description of TPCx-IoT

This section provides a detailed description of the design of TPCx-IoT, including background information for key decisions. For the full specification and kit download of TPCx-IoT, please refer to the TPC website [16].

A. Use Case: Power Substations of Electric Utility Providers

While the performance data of TPCx-IoT may be applied to any IoT installation that must ingest and persist massive amounts of data from a larger number of sensors, and provide real-time analysis of incoming data, the workload of TPCx-IoT has been granted a realistic context. It is used to model the power substations of a typical electric utility provider.

A schematic description of an electric utility provider's infrastructure is illustrated in Figure 2. It consists of *power producers*, for example, power plants and distributed energy resources (DERs), a *power grid*, and *power consumers*.

To transport power from producers to consumers, it needs to be converted to various voltages and distributed over a grid of transmission lines. Situated at every junction of this power grid are *power substations* that undertake a variety of tasks: *Step-up transmission substations* increase the voltage so large amounts of electricity can be moved efficiently over long distances. *Step-down transmission substations* do the reverse, i.e., reduce voltage as the electric power approaches its destination. *Transmission substations* connect two or more transmission lines. They contain high-voltage switches that allow transmission lines to be connected or disconnected for maintenance. *Distribution substations* transfer power from the transmission system to the distribution system of an area. In case power needs to be fed into the power grid from distributed power generation, for example, wind farms, *collector substation* are used. They allow for power flow in the opposite direction, back into the transmission grid. According to a 2017 report of the California Energy Commission [17], the power grid of the state of California includes 3,200 power substations, of which 982 are owned by one utility provider, Pacific Gas and Electric Company (PG&E) [18].

Utility providers use IoT technology in every step from thousands of edge sensors in power plants to a few sensors measuring power consumption at consumer sites. Requirements for gateways serving edge devices in each step vary dramatically. For instance, fossil fuel power plants deploy up to 4,000 sensors [19], while neighborhood smart meters only contain one. Hence, for TPCx-IoT, we decided to model sensor data generated by power substations with 200 sensors each. Power producers, consumers, and the actual transmission lines are outside of the scope of TPCx-IoT, as indicated by the non-shaded boxes in Figure 2.

The size of power substations vary depending on their purpose and how many customers they serve. For instance, the Larkin power substation in downtown San Francisco[20] measures 12,200 square feet, while the Martin power substations [21] occupies 319,000 square feet. Power substations contain various types of sensors [22]. Figure 3 illustrates four examples: *load tap changers gassing sensors* measure gas levels in load tap changes (LTC) that regulate output voltages of transformers. Gas levels can identify overheating, coking and worn contacts of an LTC. *Metalinsulatorsemiconductor (MIS) gas sensors* measure H_2 and C_2H_2 levels[23]. *Phasor measurement units* (PMU) measure electrical waves using synchrophasors [2]. *Leakage current sensors* measure the amount of current leakage to earth (ground). These sensors send their data via edge devices to a gateway that can be situated close to the power substation or in a datacenter/cloud.

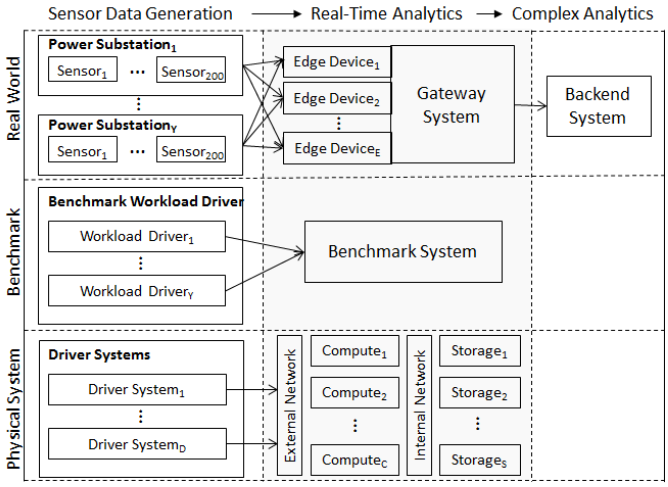

Fig. 4: Mapping of real-world, benchmark, and physical devices

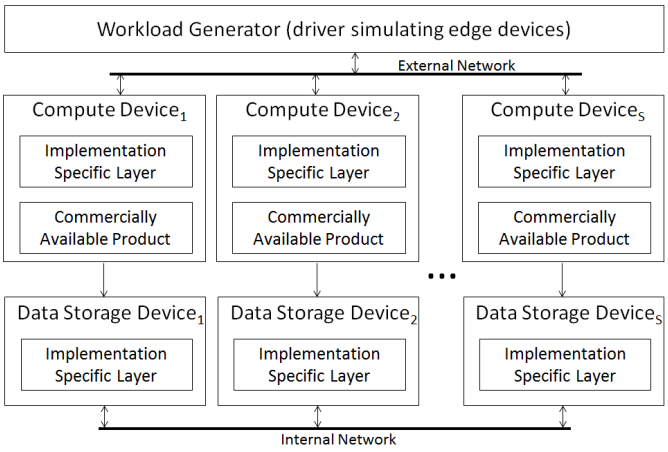

Fig. 5: TPCx-IoT system under test (SUT)

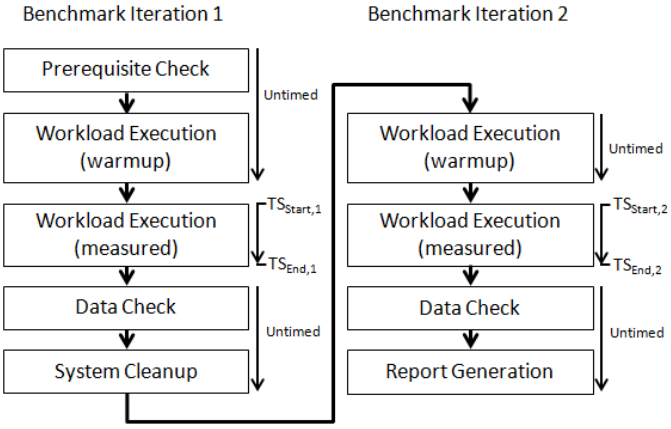

Fig. 6: Benchmark Execution Rules

The gateways ingest sensor data at a high rate, store that data for short durations, and provide real-time analytics to help take immediate action in associated power substations. Gateways send data for long-term storage and complex analytics to an analytical platform. This is, however, out of scope of TPCx-IoT.

Figure 4 shows how the real-world use case of power substations (see Figure 2) is mapped to the specification of TPCx-IoT and how it is mapped to physical systems. Each power substation in the simulated "real-world" maps directly to one instance of *TPCx-IoT workload driver* on the benchmark level. One instance of the workload driver generates sensor data from one power substation and the corresponding queries that can be run against that data. The physical level shown in the bottom of Figure 4 lists the number of physical systems needed to support the number of simulated power substations, i.e., the number of driver instances that can be run. So long as the run-time requirements for the driver are met, driver instances can be run on one or across multiple physical systems. These systems are not priced.

The gateway supporting our power substations is mapped directly to the system being tested, the *System Under Test* (SUT). SUT is a TPC term that defines all components of the system being tested. It contains priced and non-priced components. Each TPC benchmark defines the mandatory and optional parts of its SUT. TPCx-IoT defines the SUT to represent an IoT gateway system consisting of commercially available servers, switches, and storage systems running a database management system, for example, a NoSQL database. Figure 5 shows a diagram of the SUT as defined in the TPCx-IoT specification.

B. Execution Rules

Execution rules are a basic component of any benchmark definition. Because execution rules can highlight the strengths and reveal the weaknesses of products, discussions to reach an agreement about these rules are usually very controversial and long lasting in benchmark consortia.

A TPCx-IoT *benchmark run* is composed of two *benchmark iterations* (see Figure 6). Each benchmark iteration executes the TPCx-IoT *workload* twice, one to warm up the system and one for the measured execution of the workload. Before the first warmup run, the benchmark driver performs a couple of prerequisite checks: The *file check* compares the checksums (md5sum) of all non-changeable kit files[2] on the system with reference checksums in the kit; The *data replication check* makes sure that the storage system uses three way replication of data. If any of these two checks fail, the benchmark driver aborts the run.

The warmup run is not timed. The start and end timestamps of the measured run are denoted by $TS_{a,b}$ where $a \in \{start, end\}$ and $b \in \{1, 2\}$. The elapsed time of a workload execution is TE_a where $a \in \{1, 2\}$. Each *workload execution* performs concurrent data ingestion (write) and query (read) operations.

The amount of data to be ingested is a parameter to the workload driver. After the measured workload execution is completed a *data check* assures that the benchmark run fulfills

[2]Most files in the kit must not be altered by a benchmark sponsor. However configuration related files may be altered.

all necessary runtime requirements. The second benchmark iteration is a repetition run to ensure measurement repeatability. To achieve identical conditions in both runs, a *system cleanup* is performed between iterations. A cleanup consists of purging all data ingested into the data management system during the previous warmup and measured workloads, deleting all temporary files, and restarting the data management system. No additional activities are allowed between the end of the first benchmark iteration and start of the second one. After the system cleanup is complete the second iteration is executed. After data check of the second iteration, the TPCx-IoT driver runs a report that prints all information needed to audit and publish a benchmark result.

For each benchmark run the benchmark sponsor must choose the amount of ingest data and the number of simulated power substations, such that the following two execution rule requirements are fulfilled:

1) *Workload execution elapsed time*: Both the warmup and the measured workload execution need to run for at least $1800s$ each.
2) *Sensor data ingest rate*: The benchmark system must guarantee a minimal average data ingest rate per sensor of $20\frac{kvps}{s}$.

kvp stands for key-value-pair, plural is $kvps$. It is commonly used in NonSQL systems to describe a set of two data items, *key* and *value*. The key is a unique identifier for the value. In our use case kvp stand for the sensor readings arriving to the gateway system from edge devices. In the remainder of the paper we will use the terms $kvp(s)$ and sensor reading(s) interchangeably.

Each workload run must be at least $1,800s$ so that the benchmarked system can demonstrate that it can sustain high performance during an extended period before data gets forwarded to analytical systems in the back-end.

A minimal average data ingest rate per sensor is required to prevent benchmark sponsors[3] from artificially reducing the amount of reads required by queries. Assuming a fixed number of sensors there is a direct correlation between the system-wide data ingest rate per sensor and the average number of readings retrieved by each query, because each query reads data from a $5s$ interval. Keeping the system-wide data ingest rate constant, because the gateway system is saturated, one could reduce the average per sensor ingest rate by increasing the number of sensors, that is, by increasing the number of power substations/TPCx-IoT driver instances.

Requiring a minimal average ingest rate of $20\frac{kvps}{s}$ per sensor has a couple of other consequences, which we are discussing below. Because one power substation has 200 sensors, the system-wide minimal average ingest rate is:

$$200 sensors * 20\frac{kvps}{sensors * s} = 4000\frac{kvps}{s} = 3.91\frac{MB}{s} \quad (1)$$

[3]Vendors producing benchmark results and publishing them under TPC rules.

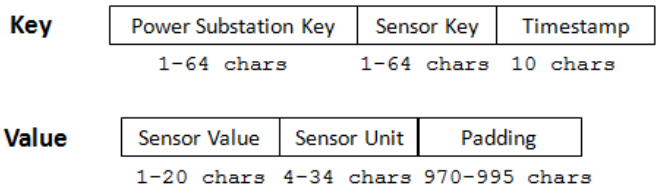

Fig. 7: Sensor reading (kvp) generated by the driver program of TPCx-IoT

The minimal average number of $kvps$ retrieved per query is:

$$20\frac{kvps}{sensor * s} * 5s = 100\frac{kvps}{sensor} \quad (2)$$

C. Data Ingestion Workload

The TPCx-IoT workload generator is based on the Yahoo! Cloud Serving Benchmark framework (YCSB) [24]. We chose YCSB for developing of TPCx-IoT because it is well-known, open source, easily adaptable to specific needs, and its built-in database interface layer allows for connections to many common open-source database management systems including NoSQL DBMS.

Before diving into the details of the changes, we briefly review the requirements of the data ingestion part of our workload driver. As outlined in the use case description (Figure 4) workload driver of TPCx-IoT must be able to generate sensor data related to different power substations. In real life, power substations vary in size depending on the number of transmission lines they connect or the number of customers they serve. The number of sensors in a substation varies accordingly. However, in our model, we assume that each substation has the same number of sensors, namely, 200.

To distinguish sensor data from different power substations, we added support in YCSB for keys and values that are based on attributes from sensors commonly deployed in power substations of utility companies. Each YCSB instance, which we refer to as a TPCx-IoT driver instance, generates sensor data arriving from one power substation. The power substation key is passed to the TPCx-IoT driver instance along with the number of sensor readings it should generate (SR). Figure 7 shows the structure of key-value pairs. Each represents one reading of one sensor of one power substation. The *key* part consists of the *power substation key*, which uniquely identifies a power substation, the *sensor key*, which uniquely identifies a sensor within a power substation and the *timestamp*, coded as *POSIX time*, which represents the time the sensor reading was taken. The *value* part consists of the *sensor value*, which represents the value that the sensor read, *sensor unit*, which represents the measurement unit of the sensor value, and *padding*, which contains random text to fill a kvp to one KByte.

Figure 8 illustrates the bare generation speed at which TPCx-IoT drivers generate $kvps$. We measure this speed by redirecting the driver's output to $/dev/null$. These experiments where conducted on a Cisco UCS C220 M4 driver system with 128GB main memory and two Intel Xeon 2680

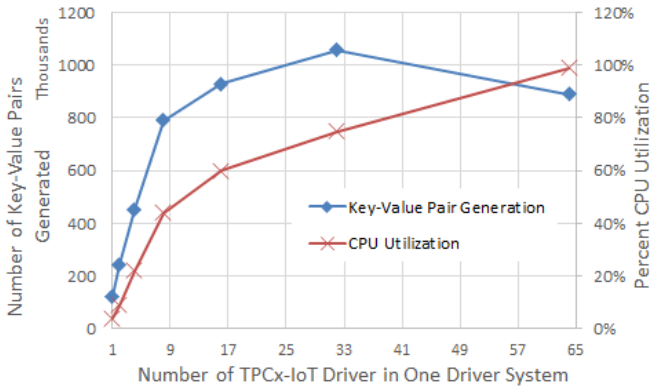

Fig. 8: Key-Value Pair Generation Speed

v4 CPUs running at 2.4 GHz. Each socket had 14 cores, 28 multi-threaded. On the x-axis, we varied the number of TPCx-IoT drivers from 1 to 64. The left y-axis shows the aggregated driver throughput in thousand $kvps$ per second ($\frac{kvps}{s}$). The right y-axis shows the CPU utilization of the driver system in percentage. As we increased the number of TPCx-IoT drivers from 1 to 64, the total throughput increased from 120,000$\frac{kvps}{s}$ with one driver to 1.1 Million$\frac{kvps}{s}$ with 32 drivers, while CPU utilization increases from 4% with one driver to 75% with 32 drivers. Total throughput drops to 900,000$\frac{kvps}{s}$ with 64 drivers with CPU utilization increasing to 100% with the system CPU portion increasing from 5% to 15%. This is not surprising because 64 drivers spawn 640 threads, which impose overheads of garbage collection and scheduling. The above generation speed of 1.1 Million$\frac{kvps}{s}$ is sufficient to simulate the amount of data generated by one power substation. Each TPCx-IoT driver simulates the data volume of one power substation with 200 sensors. That means, 1.1 Million$\frac{kvps}{s}$ generated with 32 drivers simulates a sensor frequency of about 170 samples per second (sps) per sensor.

D. Query Generation/Execution

We added support for the concurrent querying of ingested sensor data. Unlike in the traditional YCSB deployments, where YCSB picks a random set of keys to read, TPCx-IoT issues queries that read random key ranges. Queries are generated from the following four query templates, which represent typical dash-board-like queries:

1) *Max-Reading*: The max-reading query compares the maximum sensor reading in the two intervals.
2) *Min-Reading*: The min-reading query compares the minimum sensor reading in the two intervals.
3) *Average-Reading*: The average-reading query compares the average sensor reading in the two intervals.
4) *Reading-Count*: The sample-count query compares the number of sensor readings in the two intervals.

Each query compares the readings of one sensor of one power substation ingested in the last $5s$ by using the data from a randomly selected $5s$ interval from the previous $1800s$. We chose the interval of the second query to be randomly picked,

```
Scan s=new Scan();
ResultScanner scanner = null;
try {
 s.setTimeRange(timestamp,timestamp+ 5000);
 StringBuffer startKey=new StringBuffer();
 startKey.append(clientFilter);
 startKey.append(":");
 startKey.append(filter);
 startKey.append(":");
 startKey.append(timestamp);
 StringBuffer endKey=new StringBuffer();
 endKey.append(clientFilter);
 endKey.append(":");
 endKey.append(filter);
 endKey.append(":");
 endKey.append(timestamp+5000);
 s.setStartRow(startKey.toString().getBytes());
 s.setStopRow(endKey.toString().getBytes());
 if (fields==null) {
   s.addFamily(columnFamilyBytes);
 } else {
 for (String field:fields) {
   s.addColumn(columnFamilyBytes,Bytes.toBytes(field));
 }
}
scanner = currentTable.getScanner(s);
int numResults = 0;
for (Result rr = scanner.next();
      rr != null;
      rr = scanner.next()) {
  String key = Bytes.toString(rr.getRow());
  if (debug) {
    System.out.println("Got result for key: " + key);
  }
 }
}
```

Listing 1: Sample Query Code

as opposed to be from a fixed time window to minimize caching effects. All four query templates perform projections, selections, and aggregations. The projection returns the fields required by the query, namely sensor value and time stamp. The selection filters data relevant to a specific power substation, sensor, and date range. The aggregation performs max, min, average, and count operations. Listing 1 shows a sample query.

Because the database is empty before the start of the warmup run, queries issued during the warmup run might not return any data for the second, randomly chosen, time interval, because there might not exist any data in that time interval. This is not a problem as the warmup run is not timed nor is any information from it part of the metric.

E. Benchmark Driver

Figure 9 shows the architecture of the benchmark driver of TPCx-IoT. It is responsible for running the entire workload: performing prerequisite checks, performing data inserts (write operations), executing queries (read operations), checking data, performing system cleanup, and, generating reports. It serves as a wrapper around the TPCx-IoT driver instances, which are based on YCSB. Depending on the gateway size many of the TPCx-IoT driver instances are spawn.

The benchmark driver is invoked with two arguments, *number of TPCx-IoT driver instances* and *total number of* $kvps$. The number of TPCx-IoT driver instances determines how many processes generate data, how many power substations

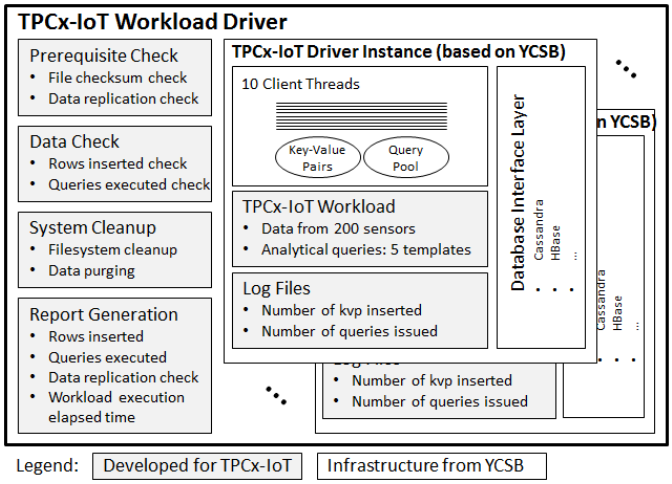

Fig. 9: Architecture of TPCx-IoT workload generator

are simulated and how many different sensors send data to the gateway. The total number of $kvps$, which simulates the number of sensor readings sent to the gateway, determines how many $kvps$ are ingested into the gateway. The default number of $kvps$ is 1 Billion. By increasing the number of $kvps$ the benchmark sponsor can adjust the run time of the workload run. Because the metric is throughput-based longer runs offer no advantages.

Each TPCx-IoT driver instances i generates about the same number of $kvps$, $KVP(i)$. With P being the number of simulated power substations and K being the total number of $kvps$ generated by all drivers, the number of $kvps$ generated by each driver can be calculated as follows:

$$KVP(i) = \begin{cases} \lfloor \frac{K}{P} \rfloor & \text{if } 1 \leq i < P \\ \lfloor \frac{K}{P} \rfloor + K \bmod P & \text{otherwise} \end{cases} \quad (3)$$

Because real benchmark configurations are expected to use values for K that are much larger (Billions) than P (hundreds), the above equation will not introduce a significant workload skew.

F. Metrics

One of the core values of the TPC is to develop benchmarks with robust, simple and verifiable performance and price performance metrics. TPC benchmark sponsors invest considerable amount of money and engineering hours to produce benchmark results. Therefore, one of the main goals of the TPC is to provide a single robust performance metric that allows for system performance comparisons over extended periods of time and, thereby, preserving benchmark investments. TPCx-IoT defines three primary metrics:

1) Performance metric: IoTps;
2) Price-performance metric: $/IoTps; and
3) System availability: Date.

The performance metric reflects the effective gateway ingestion rate in seconds that the SUT can support during a 30 minute measurement interval, while executing real-time analytic queries. The total number of $kvps$ ingested into the database, N_i, where $i \in \{1, 2\}$ being the iteration, is used to calculate the performance metric. The performance run is defined as the measured run m with the lower number of $kvps$ ingested, i.e., $m \in \{1, 2\}, n \in \{1, 2\}$ such that $m \neq n$ and $N_m < N_n$. IoTps is calculated as follows:

$$IoTps = \frac{N_m}{TS_{end,m} - TS_{start,m}} \quad (4)$$

The price-performance metric reflects the total cost of ownership per unit IoTps performance. It is calculated as follows:

$$\frac{\$}{IoTps} = \frac{ownership\ cost * (TS_{end,m} - TS_{start,m})}{N_m} \quad (5)$$

The system availability metric reflects the date when all line items of the price configuration are generally available,that is, to any costumer. The system availability metric is important because it guarantees that the benchmarked system is a production system that can be purchased and not an experimental system.

IV. Applying TPC Benchmark Model

Formal specifications for data generation, workload generation and benchmark execution are key for defining a good benchmark. However, industry standard benchmarks have additional requirements. They need to ensure that the benchmarked systems use realistic configurations and are priced in a fair and consistent manner. Moreover, they need to ensure that all benchmark rules are followed, and benchmark execution is documented properly. The TPC has developed benchmark models and rules that aid in turning a benchmark into an industry standard benchmark.

A. About TPC Benchmark Classes and Why Express for IoT

A *benchmark class* in TPC is a set of benchmark standards that share the same characteristics and the same rules for creation, maintenance, and publication. The TPC currently has two benchmark classes, namely, *enterprise* and *express*. Enterprise class benchmarks, namely, TPC-C, TPC-E, TPC-H, TPC-DS, and TPC-DI, are defined in a technology agnostic way in the form of paper specifications. They can be implemented using any technology that fulfills the requirements defined in their respective specifications. This model has served the TPC well for many years because it accommodates differences hardware architecture and software functions and features. The last decade has seen a shift toward the construction of enterprise systems based on commodity hardware, especially, x86-based systems and open-source software. This convergence to fewer architectures has weakened the need for a technology-agnostic specification. This led the TPC to develop the express benchmark framework. Express benchmarks are based on predefined executable software kits. They are restricted to the technology implemented in the kit. Consequently, they can be deployed rapidly with minimal modifications, while satisfying the rigid rules TPC benchmarks are known for. There are many pros and cons in choosing a benchmark class for a given

benchmark. For a detailed discussion see [25]. TPC currently supports four express benchmarks, TPCx-BB, TPCx-V, TPCx-HS, and TPCx-IoT.

Regardless of class, all benchmarks developed by the TPC share the same values of the TPC, namely:

- publish only one performance metric,
- publish a price/performance metric,
- ensure availability of the measured system,
- ensure repeatability of a benchmark run and
- rigid audit process

We selected the express model for TPCx-IoT because gateway architectures are mostly based on industry standard servers (x86-based) and use open software stacks, which makes the development of a kit feasible. Development times for express benchmarks are much shorter compared to those for enterprise benchmarks.

B. Priced Configuration

The TPC is one of the few performance consortia that defines very strict rules for pricing a system and defining availability of components of a system. Karl Huppler from IBM has summarized the pricing methodology and outlined future direction of TPC pricing in [26].

Rules for pricing the *priced system* and its associated software and maintenance are included in the TPC *pricing specification* [27], which is listed under common benchmark specifications. Common benchmark specifications apply to all TPC benchmarks.

The system to be priced must include all hardware and software components listed in the SUT, a communication interface that can support user interface devices, additional operational components configured on the test system, and maintenance of all above components. The cost of a priced system consists of the following costs:

- SUT price;
- Price of additional products (software or hardware) in system;
- Price of additional products (software or hardware) required for customary operation, administration and maintenance of the SUT for three years;
- Price of all products required to create, execute, administer and maintain the executables necessary to create and populate the test environment.

The following components are excluded from the cost of a priced system:

- End-user communication devices and related cables, connectors, and switches and
- Equipment and tools used exclusively for FDR[4] production.

Pricing is strictly associated with the availability of system components. If any component of a benchmark becomes unavailable, the benchmark result must be withdrawn. If, however, components with comparable performance are available, the TPCx-IoT pricing rules allow for component substitutions. If corrections to components of the priced configuration are required during the life of a product, these changes are not considered substitutions as long as the part number of the priced component is identical to the new component. The idea behind this change is that hardware and software suppliers may update the components of the priced configuration so long as these updates do not negatively impact the reported performance metric or numerical quantities by more than two percent.

The following are not considered substitutions:

1) Software patches to resolve a security vulnerability;
2) Silicon revision to correct errors;
3) New supplier of functionally equivalent components, for example, memory chips and disk drives.
4) Durable Media (for example, disk drives) and cables[5]

C. Full Disclosure Report and Executive Summary

Each benchmark results is required to provide a *full disclosure report* (FDR) and *executive summary* (ES). The intent of these disclosures is to simplify comparisons between results and for a facilitating replication of the results of any given benchmark, given appropriate documentation and products. The FDR must contain all customer-tunable parameters and options that have been changed from the defaults found in actual products, including but not limited to the configuration parameters and options of the operating system, server, storage, network, and any other hardware components incorporated into the pricing structure. In addition if any software is specifically compiled for running TPCx-IoT, any compiler optimization options must be disclosed.

Furthermore, the FDR must list diagrams of both the measured and the priced configurations, along with by a description of the differences. The diagrams must clearly show the number of nodes used and the total number and type of processors used including sizes of L2 and L3 caches. The diagrams must further show the size of memory; any specific mapping/partitioning of memory that is unique to the SUT; number and type of disk units (and controllers, if applicable), number of channels or bus connections to disk units, including protocol type, number and speed of LAN connections and switches; and any other hardware components physically used in the test or incorporated into the pricing structure.

D. Audit Requirements

Before publishing any TPCx-IoT result, the data must be audited by an independent audit or by peer audit. The benchmark sponsor may choose between the two audit methods. An independent audit is conducted by a third party with no interest in the benchmark sponsor, while a peer audit is defined as the process of reviewing benchmark results by a peer review committee consisting of three members from TPC companies other than the benchmark sponsor.

[4]Full Disclosure Report

[5]A durable medium is defined as a data storage medium that is inherently non-volatile, such as a magnetic disk or tape

V. Experiments

We ran our tests against HBase 1.2.0 [28], [29], which is a good representative system for this workload. Key-value stores for IoT data are being used in a variety of products and services, such as Google Cloud IOT [30] and Microsoft Azure IoT [31]. Our benchmarks are run on a cluster of eight Cisco UCSB-B200-M4 blade servers, each with the following configuration:

- 2 Intel(R) Xeon(R) CPU E5-2680 v4 clocked at 2.40GHz, each with 14 cores and 28 threads.
- 256 GB RAM
- 2 Cisco UCS 6324 FI (10 GBps per server node)
- 2 Samsung 3.8 TB 2.5-inch Enterprise Value 6G SATA SSD

We used the following tuning parameters for HBase:

- hbase.client.write.buffer=8GB
- hbase.regionserver.handler.count=224
- Maximum number of Write-Ahead Log (WAL) files=128
- hbase.hstore.blockingStoreFiles=28
- Java Heap Size of HBase RegionServer=32GB
- Client Java Heap Size=8GB

This section is intended to show basic performance characteristics of TPCx-IoT. It is not intended to showcase the best performance of HBase nor to analyze HBase in detail. The above tuning steps for HBase follow best practices.

A. Scaling the Number of Power Substations

In order to determine how many power substations our gateway system supports, while still fulfilling the execution rule requirements outlined in Section III-B, we ran experiments varying the number of power substations (number of TPCx-IoT instances) from 1 to 48. We increased the number of power substations in steps of power of 2, except from 32-48, where we added 16.

TABLE I: Experiment Parameters & Requirement Fulfillment

Power sub-stations	Rows Ingested [Million]	Elapsed Time [s]		Ingestion Rate $\frac{kvps}{s}$	
		Warm-up	Mea-sured	System-Wide	Per-Sensor
1	50	4,795	5,099	9,806	49.0
2	60	2,024	2,222	26,999	67.5
4	100	1,813	1,812	56,822	71.0
8	240	2,606	2,837	84,602	52.9
16	400	2,822	2,986	133,940	41.9
32	400	1,897	2,149	186,109	29.1
48	400	1,992	2,188	182,815	19.0

Table I lists the benchmark input parameters, number of power-substations and number of $kvps$ to be ingested, in the first two columns. As we increased the number of power substations from 1 to 48, we increased the number of rows ingested from 50 Million to 400 Million to keep the elapsed times of the warmup and measured executions of the workload larger than 1800 s (columns three and four). Finding a suitable number of rows to ingest was not difficult. We binary-searched a suitable number for one power substation and extrapolated

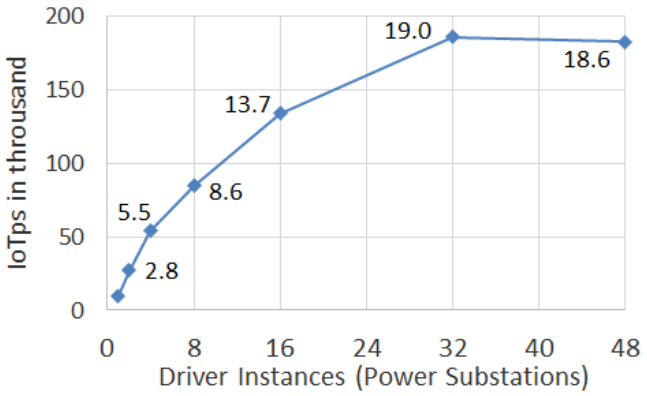

Fig. 10: System-wide kvp inserted per second [IoTps]

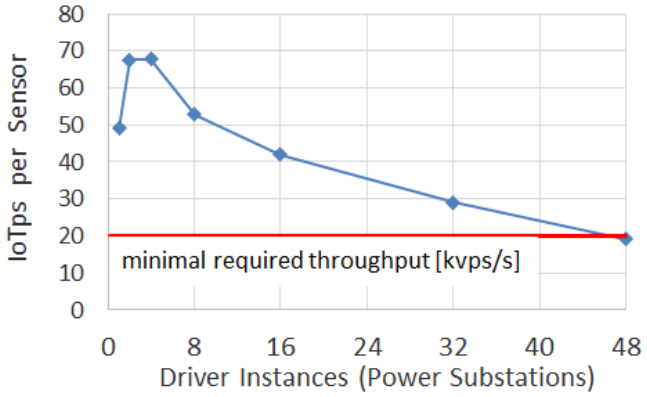

Fig. 11: Per Sensor kvp inserted per second

to multiple power substations. The system-wide throughput, which corresponds to the main metric ($IoTps$) of TPCx-IoT, increases from 9,806 $\frac{kvps}{s}$ with one power substation to 186,109 $\frac{kvps}{s}$ with 32 power substations. Adding 16 more power substations keeps the throughput constant at about 182,815 $\frac{kvps}{s}$. The per-sensor ingestion rate requirement of $20\frac{kvps}{s}$ is fulfilled up to 32 power substations. At 48 substations it falls to $19\frac{kvps}{s}$.

The following graphs show additional metrics of the above conducted experiments. Unless noted otherwise, the number presented are measured, not derived numbers. Figure 10 plots the system-wide throughput ($IoTps$). For each data point we display the scaling number s based on the throughput with one power substations, that is, $S_i = \frac{IoTps_i}{IoTps_1}$ with $i \in 2, 4, 8, 16, 32, 48$ being the number of power substations. $IoTps$ scales super-linear until eight power substations. With two power substations S_2 is 2.8, with 4 power substations S_4 is 5.5 and with 8 power substations S_8 is 8.6. With 16 and more power substations $IoTps$ scales sub-linearly at $S_{16} = 13.7$, $S_{32} = 19.0$ and $S_{48} = 18.6$.

Figure 11 plots the measured average per-sensor $IoTps$. The red line at 20 $IoTps$ indicates the minimum allowed for a valid benchmark run. $IoTps$ increases from 49.0 $IoTps$ with one power substation to 67.8 $IoTps$ with 4 power substations.

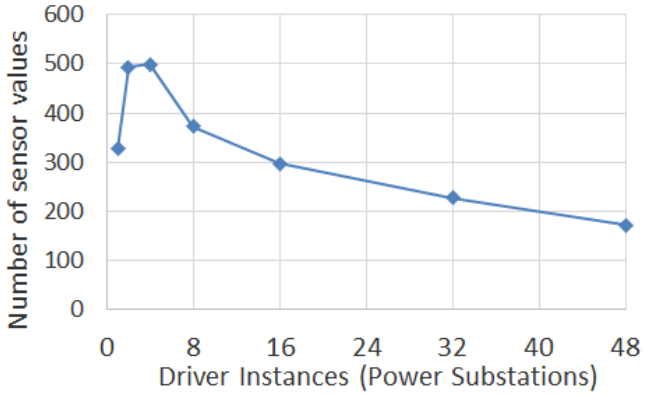

Fig. 12: Average sensor readings aggregated per query

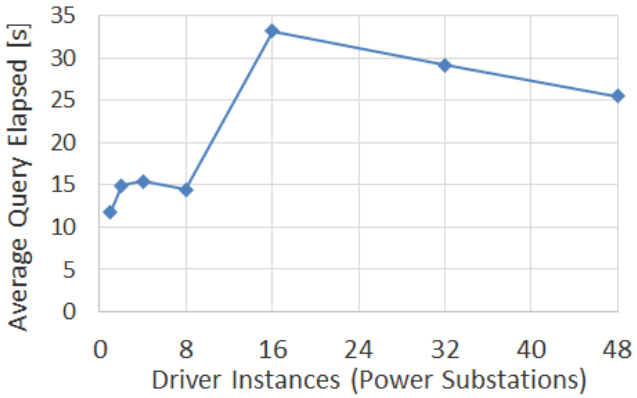

Fig. 13: Average system-wide query elapsed time [s]

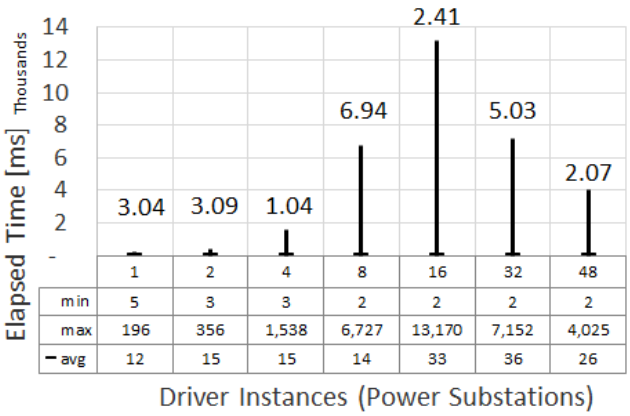

Fig. 14: Query elapsed time variation

Increasing the number of power substations to 48 causes $IoTps$ to drop below the allowable limit. The reason for the initial increase and subsequent drop in per-sensor throughput is the initial super-linear system-wide throughput scaling, which turns into a sub-linear scaling with 16 power substations. Because the throughput per sensor is calculated by dividing the system-wide throughput by the number of sensors and because the number of sensors increases linearly with the number of power substations, there is a direct correlation between the system-wide scaling and the per-sensor throughput: Sub-linear system-wide scaling causes a decrease in per-sensor throughput, linear system-wide scaling causes constant per-sensor throughput, and super-linear system-wide scaling causes an increase in per-sensor throughput.

Figure 12 plots the measured average number of $kvps$ aggregated by each query. This is essentially the average number of readings read per query to calculate the dash-board value for each sensor. The curve looks very similar to that of the previous Figure 11, because the number of data aggregated as the same correlation as the $Iotps$ per sensor. We include this graph to show that a reasonable number of values is considered to calculate the aggregate for each of the query template. If the number drops below 200, the benchmark run is invalid.

Figure 13 plots the average system-wide query elapsed time for each of our runs. Average query elapsed time is betweem 11.8 ms and 14.4 ms with up to 8 power substations. With 16 power substations average query elapsed time increases to 33.1 ms and reduces slightly to 29.1 ms with 32 power substations and 25.4 ms with 48 respectively.

The bar chart in Figure 14 shows minimum, maximum and average query elapsed times in milliseconds. For each number of power substations the minimum and average query elapsed times are in the double digit milliseconds (max is 36 milliseconds). However, the maximum query elapsed times are very high starting with 4 power substations (larger than 1000 ms). The numbers above each bar show the coefficient of variation ($\frac{stdev}{mean}$). For each of the runs the coefficient of variation is larger than 1, which indicates a very large variation. We also calculated the 95 percentiles for each run (not shown in figure). They are below 25 ms up to 16 power substations. They increase to 185 ms with 32 and 143 ms with 48 power substations. The benchmark currently does not require a query elapsed time percentile. However, because of the dash-board like use-case of TPCx-IoT, this is something to consider.

Because TPCx-IoT requires the complete ingestion of a fixed number of sensor readings $kvps$ from each power substation, as opposed to run for a fix amount of time, the

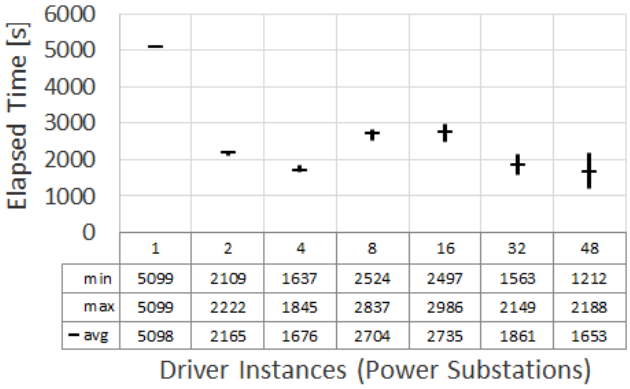

Fig. 15: Power substation workload variation

benchmark tests the a gateway system's ability to load-balance the data ingestion between all power substations. Figure 15 plots the minimum, maximum, and average ingestion times for data from all power substation (see Table II for a listing of the numbers). Although with one power substation all three values are identical, we include them for completeness. As we increase the number of power substations the difference between the fastest and the slowest ingest time increases from 5% to 81%. Table II shows the times for the fastest ingest in the second column, the slowest ingest time in the third column and the average ingest time in the fourth column for each of the runs. The difference between the fastest and slowest are printed in Column 4 (absolute) and Column 5 (relative). For instance, the large difference of 81% in the 48 power substation case indicates that there is large potential for speeding this run up.

TABLE II: Difference between fastest and slowest power substation ingest time

Power substation	Ingest Time [s]			Difference	
	Min	Max	Avg	Absolute	Relative
1	5,099	5,099	5,099	n.a.	n.a.
2	2,109	2,222	2165	113s	5%
4	1,637	1,845	1,676	208s	13%
8	2,524	2,837	2,704	313s	12%
16	2,497	2,986	2,735	489s	20%
32	1,563	2,149	1,861	586s	38%
48	1,212	2,188	1,653	976s	81%

B. Scaling the Number of Gateway Nodes (Scale-Out)

For the following set of experiments we scaled the number of HBase servers from two to eight nodes. Due to the replication requirements in TPCx-IOT the minimum number of nodes eligible for benchmark publication is two. All experiments conducted in this section fulfilled the runtime requirement of a minimal workload execution time of 1,800 s.

Figure 16 shows how TPCx-IoT throughput [$IoTps$] scales with the number of power substations on all three configurations. The corresponding data is also listed in Table III. With the 2-node configuration (blue graph with rhombus markers) system-wide throughput increases from 21,909 $IoTps$ with one power substation to 105,877 $IoTps$ with eight power substations. Increasing the number of substations beyond eight increases system-wide throughput to a peak of 115,486 $IoTps$. The 4-node configuration shows 15,706 $IoTps$ with one power substation (red graph with square markers). This is about 28% less throughput than the two-node configuration. System-wide throughput increases to 125,603 $IoTps$ with 16 power substations. Increasing the number of power substations beyond 16 increases system-wide throughput to peak at about 134,248 $IoTps$. The 4-node configuration delivers a 16% higher peak system-wide throughput. As discussed in Section V-A the 8-node configuration scales until 32 power substations, after which performance remains flat. At 9,806 $IoTps$ the 8-node configuration delivers the lowest performance with one power substations. However, with 182,815 $IoTps$ it is able to deliver the highest peak system-wide throughput.

TABLE III: System-wide and per-sensor throughput for 2,4 and 8 nodes

Power sub-stations	Throughput [IoTps]					
	System-wide			Per-sensor		
	2-node	4-node	8-node	2-node	4-node	8-node
1	21,909	15,706	9,806	109.5	78.5	49.0
2	38,939	33,612	26,999	97.3	84.0	67.5
4	63,076	57,113	54,201	78.8	71.4	67.8
8	105,877	90,160	84,602	66.2	56.4	52.9
16	114,508	125,603	133,940	35.8	39.3	41.9
32	114,764	132,100	186,109	17.9	20.6	29.1
48	115,486	134,248	182,815	12.0	14.0	19.0

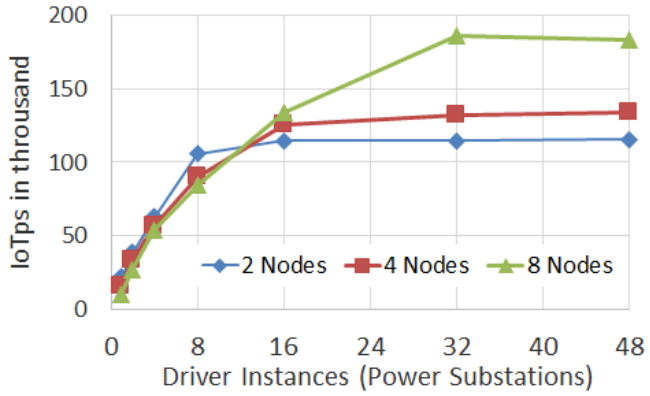

Fig. 16: System-wide kvp inserted per second [IoTps]

Figure 17 shows the per-sensor throughput for the three configurations. Similarly to the 8-node configuration, the 4-node configuration shows an initial increase in per-sensor throughput from one to four power substations. Per-sensor throughput decreases after that. It stays above the limit of 20 $IoTps$ until shortly after 32 power substations. The 2-node configuration does not show an increase in per-sensor throughput from one to two power substations, because, unlike the other configurations, the 2-node configurations shows sub-linear scaling starting from one to two power substations. Per-sensor throughput decreases steadily starting from one power substation to 48 power substations. It crosses the limit of 20 $IoTps$ exactly at 32 power substations.

VI. Conclusion

In this paper, we presented TPCx-IoT, the first industry standard benchmark for measuring the performance of IoT gateway systems. Similar to previous TPC express benchmarks, TPCx-IoT is a kit-based benchmark modeled after the real-world scenario of a power utility provider that must distribute power through a smart grid with many power substations. The target systems of the benchmark are gateway systems that can ingest data from edge-devices at a high speed while performing real-time analytic queries that feed a dashboard for monitoring purposes. The TPCx-IoT benchmark was accepted by the TPC in May 2017.

We presented performance results obtained from three dif-

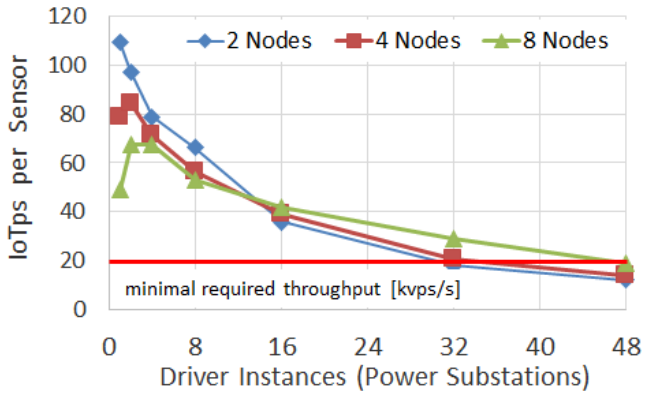

Fig. 17: Per Sensor kvp inserted per second

ferent industry configurations running HBase 1.2.0. The first set of performance experiments where we scaled the number of power substations show that the read and write operations of TPCx-IoT impose a significant workload to show system limits. It also showed that the execution rule requirement of executing the workload for at least 1,800s were easy to fulfill. The second requirement of performing at least 20 $\frac{kvps}{s}$ per sensors showed to be reasonable and is also a good gatekeeper to prevent benchmark sponsors to reduce the read requirements during query execution. The larger the configuration the more gateways could be supported. We also showed that TPCx-IoTs fixed workload requirement, that is, to ingest a fixed number of $kvps$ per power substation, can reveal deficiencies in system's ability to load balance data ingestion across multiple power substations.

Acknowledgment

The authors would like to thank all members of the TPCx-IoT subcommittee for their valuable input. This work was partially supported by the the German Ministry for Education and Research as BBDC (01IS14013A).

References

[1] Gartner, "IoT Growth Report By Gartner," http://www.gartner.com/newsroom/id/3598917, 2017, smartMeter.

[2] M. Adamiak, B. Kasztenny, and W. Premerlani, "Synchrophasors: definition, measurement, and application," *Proceedings of the 59th Annual Georgia Tech Protective Relaying, Atlanta, GA*, pp. 27–29, 2005.

[3] Bluethooth, "Core Version 5.0," https://www.bluetooth.com/ specifications/ bluetooth-core-specification, 2017, bluetooth Standard.

[4] IEEE, "802.15.4-2011 - IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)," http://standards.ieee.org/findstds/standard/802.15.4-2011.html, 2017.

[5] ——, "IEEE 802.11TM WIRELESS LOCAL AREA NETWORKS," http://www.ieee802.org/11/, 2017.

[6] E. T. S. I. (ETSI), "Mobile technologies GSM," http://www.etsi.org/technologies-clusters/technologies/mobile/gsm, 2017.

[7] OASIS, "MQTT Version 3.1.1 Plus Errata 01," http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html, 2017, mQTT Standard.

[8] TPC, "Home Page TPC," http://www.tpc.org, 2017.

[9] ——, "TPCx-IoT first standard for IOT Gateway systems," http://www.tpc.org/tpcx-iot/default.asp, 2017.

[10] "IoTMARK," http://www.eembc.org/iot-connect/about.php, 2017, eMBC.

[11] M. F. Arlitt, M. Marwah, G. Bellala, A. Shah, J. Healey, and B. Vandiver, "Iotabench: an internet of things analytics benchmark," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, Austin, TX, USA, January 31 - February 4, 2015*, L. K. John, C. U. Smith, K. Sachs, and C. M. Lladó, Eds. ACM, 2015, pp. 133–144. [Online]. Available: http://doi.acm.org/10.1145/2668930.2688055

[12] "Smart Meter," http://en.wikipedia.org/wiki/Smart_meter, 2017, smartMeter.

[13] M. Poess and C. Floyd, "New TPC Benchmarks for Decision Support and Web Commerce," *SIGMOD Record*, vol. 29, no. 4, pp. 64–71, 2000.

[14] R. O. Nambiar and M. Poess, "The Making of TPC-DS," in *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, 2006, pp. 1049–1058.

[15] M. Pöss, R. O. Nambiar, and D. Walrath, "Why You Should Run TPC-DS: A Workload Analysis," in *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, 2007, pp. 1138–1149.

[16] TPC, "TPC EXPRESS BENCHMARKTM IoT (TPCx-IoT) Standard Specification Version 1.0.0," http://www.tpc.org/TPC_Documents_Current_VersionspdfTPCx-IoT_v1.5.x.pdf, 2017, tPCx-IoT specification.

[17] C. E. Commission, "Energy Infrastructure: Statewide Operational Substations," http://www.energy.ca.gov/ maps/ powerplants/ Operational_ Substations.xlsx, 2017.

[18] P. Gas and Electric, "Pacific Gas and Electric Company Company Profile," https://www.pge.com/en_US/about-pge/company-information/profile/profile.page, 2017.

[19] S. Selvam, "Efficient monitoring in fossil-fueled power plants using low-cost wireless sensors," 2015, pp. 2319–3344.

[20] tefDesign, "PG&E Larkin Substation," http://tefarch.com/projects/detail/64/, 2017.

[21] S. F. P. Department, "320400 Paul Avenue Data Center and associated Extension of PG&E 12kV Electrical Distribution Circuits," http://sfmea.sfplanning.org/2011.0408E_FMND.pdf, 2014.

[22] E. P. R. Institute, "Sensor Technologies for a Smart Transmission System," http://www.remotemagazine.com/images/EPRI-WP.pdf, 2009.

[23] G. F. Fine, L. M. Cavanagh, A. Afonja, and R. Binions, "Metal oxide semi-conductor gas sensors in environmental monitoring," *Sensors*, vol. 10, no. 6, pp. 5469–5502, 2010. [Online]. Available: http://www.mdpi.com/1424-8220/10/6/5469

[24] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010*, J. M. Hellerstein, S. Chaudhuri, and M. Rosenblum, Eds. ACM, 2010, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807152

[25] K. Huppler and D. Johnson, "TPC express - A new path for TPC benchmarks," in *Performance Characterization and Benchmarking - 5th TPC Technology Conference, TPCTC 2013, Trento, Italy, August 26, 2013, Revised Selected Papers*, 2013, pp. 48–60.

[26] K. Huppler, "Price and the TPC," in *Performance Evaluation, Measurement and Characterization of Complex Systems - Second TPC Technology Conference, TPCTC 2010, Singapore, September 13-17, 2010. Revised Selected Papers*, ser. Lecture Notes in Computer Science, R. O. Nambiar and M. Poess, Eds., vol. 6417. Springer, 2010, pp. 73–84. [Online]. Available: https://doi.org/10.1007/978-3-642-18206-8_6

[27] TPC, "TPC Pricing Specification Version 2.1.1," http://www.tpc.org/ TPC_ Documents_ Current_ Versions /pdf/ Pricing_ v2.1.1.pdf.

[28] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data (awarded best paper!)," in *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*, B. N. Bershad and J. C. Mogul, Eds. USENIX Association, 2006, pp. 205–218. [Online]. Available: http://www.usenix.org/events/osdi06/tech/chang.html

[29] Apache, "Apache HBase homepage," https://hbase.apache.org/, 2017.

[30] Google, "Overview of Internet of Things," https://cloud.google.com/solutions/iot-overview.

[31] Microsoft, "Overview of Azure IoT Suite," https://docs.microsoft.com/en-us/azure/iot-suite/iot-suite-overview.