Synthetic Data Generation for the Internet of Things

Jason W. Anderson, K. E. Kennedy, Linh B. Ngo, Andre Luckow, Amy W. Apon
School of Computing, Clemson University, Clemson, SC
{jwa2,kkenned,lngo,andreluckow,aapon}@clemson.edu

***Abstract*—The concept of Internet of Things (IoT) is rapidly moving from a vision to being pervasive in our everyday lives. This can be observed in the integration of connected sensors from a multitude of devices such as mobile phones, healthcare equipment, and vehicles. There is a need for the development of infrastructure support and analytical tools to handle IoT data, which are naturally big and complex. But, research on IoT data can be constrained by concerns about the release of privately owned data. In this paper, we present the design and implementation results of a synthetic IoT data generation framework. The framework enables research on synthetic data that exhibit the complex characteristics of original data without compromising proprietary information and personal privacy.**

I. Introduction

The arrival of the age of the Internet of Things (IoT) [1] creates opportunities for industry. Raw data are being collected at an increasing rate from a multitude of devices and sensors – sources that include connected homes, smart meters, manufacturing, healthcare, fitness trackers, mobile devices, vehicles, and more. The ingestion and integration of raw data, the extraction of valuable business information from the raw data, and planning for infrastructure capacity and analytic capability for analyzing this data are new challenges in the era of big data [2]. Unlike structured transactional data that follow predefined schemas and can be stored in two-dimensional tables, IoT data typically do not have a standard structure, are often represented in a hierarchical structure using XML or JSON [3, 4], and are dynamic and self-describing [5].

A combination of business and technical reasons makes addressing the challenges of IoT data difficult from within an enterprise computing environment. The current state-of-the-art is that there is no common, comprehensive solution for data wrangling problems as each company has its own unique sets of data with unique attributes. Development of a robust data infrastructure that will meet production requirements needs experimental analysis at a realistic scale. However, the budget for hardware infrastructure is typically just enough to support production Service Level Agreements and to provide the stability required to avoid failures in data operations [6]. It is often not feasible from a business perspective to provide sufficient computing resources for experimental research and development on big data.

One solution for experimental research on IoT data is to utilize academic partners or commercial cloud providers who are equipped with large scale computational resources. However, much of the data of interest are proprietary or have privacy constraints and cannot be transferred to locations outside of company resources without significant risk to the business. The issue in sharing data lies with the possibility that sensitive or proprietary data could be leaked.

An alternative approach for experimental research on realistic data sets is to create synthetic data that have characteristics that are similar to the original data but have values that are not obtained by direct measurement. This approach allows for research on enterprise data infrastructure to be performed on resources that are external to the company while protecting sensitive information. The challenges in synthetic data generation include capturing the aggregated structural and statistical characteristics of the original data that are sufficient to meet the goals of the experimental research while at the same time not revealing personally identifying information contained in combinations of the original data sets. Researchers must consider the tradeoffs between the level of statistical accuracy of the generated data, the need to maintain anonymity of the original data in the generated set, and the computational complexity and run time of the data generation process.

In this paper, we present our research on the development and implementation of a synthetic IoT data generation framework that is capable of generating terabytes of structurally similar synthetic data from a highly complex and nested original data source. Our primary motivation for developing the framework is to enable the design, development, and testing of large scale data analytic tools and data infrastructures to support IoT research. We evaluate our approach on a real world data source that includes data that have been collected over a period of several months from hundreds of sensors on millions of electronic objects located at geographically different locations. We report on the results of our efforts in two ways. We compare the stuctural characteristics of the generated data to the original data, and we evaluate the performance of a data access framework on both the original and synthetic data.

II. Background and Related Work

As a rapidly developing paradigm, the Internet of Things (IoT) presents the concept that all of the things surrounding us can communicate and exchange information via the Internet, and by doing so, they enable "autonomic responses to challenging scenarios without human intervention" [4]. However, as IoT data are extremely heterogeneous, noisy, and large-scale, this presents a major challenge to the process of cleaning, integrating, and processing the data.

Synthetic data generation is typically accomplished by developing statistical distributions for a set of samples from data that were directly measured, and then creating new values in

978-1-4799-5666-1/14/$31.00 ©2014 IEEE

the same format as the real data from these distributions. To characterize the related approaches in the problem of large scale synthetic data generation, we categorize the data as one of two types: structured data (e. g., tabular format) and semi/unstructured data (e. g., XML/JSON format).

For structured data, well-known sources of synthetic data are the two industrial general-purpose database benchmarks, TPC-C and TPC-H [7]. These benchmarks can generate arbitrary amounts of data. The generated data are inserted into flat tables with a number of predefined columns. Similar general-purpose benchmarks, which also generate synthetic data at scale, are BigBench [8] and YCSB [9].

Semi/unstructured data require a more complex approach for the generation process. The work by Aboulnaga et. al. [10] utilizes the Markovian structures of all paths from the root to every possible leaf of the XML tree to generate the synthetic data. This work does not evaluate how structurally similar the synthetic data are compared to the original data. The work by Cohen [11] enables users to generate synthetic XML documents, but requires users to provide a target DTD (document type definition) document and detailed global and local constraints on the output synthetic XML structures.

These approaches require *a priori* knowledge of the detailed foundational XML template for the original data. Today's IoT data sources exhibit a very large base XML template with millions of possible XML paths, and prior knowledge of the XML template is often not available. The goal of this research is to design and implement a scalable system for creating synthetic XML that captures the complexity and variety of presented IoT sources. Our approach is not constrained to XML data and can also be applied to other hierarchical data formats.

III. Characterization of IoT Data

The IoT synthetic data generation proceeds in two major phases. The first phase is to perform structure and value extraction from the original XML, resulting in a data characterization called the *synthesis set*. The second phase uses the synthesis set as input to generate and output synthetic XML data. In this section we present the steps of phase one, structure and value extraction.

To begin, we provide a formal definition of attributes (as in columns of a data table and not the XML attribute construct) of a data object stored as an XML document. The text contents of an XML document are the collected measurements of the sensors on an object. Each of them represents a unique attribute of that object. The term *value* is used instead of *attribute* in order to distinguish an object's data attributes from its XML attribute construct and *path* to refer to the list of opening tags and attributes that categorize the value. The name of an object attribute is determined by the path from the root tag to the first opening tag before the text content of that attribute. The path includes all opening tags and their accompanying XML attribute constructs but excludes the closing tags. For example, there are four attributes as shown in the first column of Table II for the object *device 0001* stored in the following XML:

```
<?xml version="1.0"?>
<device id="0001">
  <sensor name="a">
    <type>module 01</type>
    <weight>4.0</weight>
  </sensor>
  <sensor name="b">
    <type>module 02</type>
    <temperature>60</temperature>
</device>
```

Data characterization is based on a few assumptions. First, due to the absence of strict schemas, we assume that XML elements at the same nesting level are unordered since order is not a requirement for well-formed XML [12]. Secondly, we assume that tag attributes are an identifying characteristic of a path in the XML tree and consider paths with variations in attribute values to be distinct. We also assume that values in the XML tree can be classified as either numeric or categorical, as described below.

Data characterization includes: 1) extracting the structure of the data so that similar documents can be generated, and 2) characterizing the values that exist within the dataset. Figure 1 outlines the process. First, the values are removed and the tag/attribute trees are stored along with an MD5 hash and their frequency of occurrence into the *structure_data* file. The original XML is examined again, extracting each value along with an MD5 hash of the identifying path and the number of occurrences of that path/value combination. Non-numeric and infrequently occurring values are classified into the *categorical_values* file. The remaining values are classified as numeric for distribution fitting. The results are stored in *numeric_distributions*. The three resulting files make up the synthesis set. The condensed nature of this set also facilitates obscuring details of the data through string substitution, as will be discussed in Section IV. Next, we describe the details of these steps.

A. Structure

The first step of data characterization is to extract the structure of the data. We view the XML structure as a tree of tags along with their associated attributes but which is separate from the document's values. We use Hadoop MapReduce [13] to ingest the XML-based IoT data. The MapReduce framework processes each XML document individually. The initial map phase removes all of the node values of an XML document while leaving the tags and attributes. Also in the map phase, the MD5 hash of the remaining XML string is calculated. This hash and the XML structure become the key/value pair output of the map phase. The reduce phase aggregates these pairs and emits a list of unique triples whose first value is the MD5 hash, the second value is the frequency of this hash in the entire document set, and the third value is the stripped XML structure. Table I is an example of the output from this process, which we refer to as *structure_data* within the synthesis set.

TABLE I: Examples of the *structure_data* table.

Structure Hash	Freq.	Structure
9641ABEF...	1	<device id="0001"><sensor name="a"><type></type><weight></weight></sensor></device>
C5D7CA98...	3	<device id="0001"><sensor name="b"><type></type><temperature></temperature></sensor></device>

TABLE II: Values and the identifying MD5 hash of their path, along with the frequency of which that pair was encountered.

Tag	Path Hash	Frequency	Value
<device id="0001"><sensor name="a"><type>	B3EE890F...	1	module 01
<device id="0001"><sensor name="a"><weight>	0E742374...	1	4.0
<device id="0001"><sensor name="b"><type>	57518DAC...	3	module 02, module 03, module 04
<device id="0001"><sensor name="b"><temperature>	15FDF201...	3	60, 65, 69

The entire XML structure is stored, retaining the hierarchical characteristics of the XML documents.

B. Values

The next step in data characterization is the calculation of the statistical distributions of the different values of the data. For a tabular data set, the set of values to be considered would be the data columns, but there is no predetermined set of values for hierarchical and schema-less IoT data. The distributions of the data values must be computed across tens of millions of IoT data entries, each of which has different combinations of sensor measurements.

The values of an XML object are encoded within the path of the object's XML structure. Hadoop MapReduce is used to extract the paths into an intermediate form for classification. A recursive algorithm is used to descend the XML tree, appending each node's name and attribute list to a string. If a text or numeric value is encountered, an MD5 hash of the path string is generated and emitted with the value. In the reduce phase of the job the hash-value combinations are counted and output as the intermediate *value_data* file. For example, the XML paths and their respective hashes, reduced counts, and values of the example XML structures in Table I are shown in Table II.

The *value_data* is the input into the next phase, where the contents of the values are classified as numeric or categorical. Any set of contents for which there are 30 or fewer unique values is classified as categorical. In this initial design and implementation we make the assumption that the values are independent of each other for the purpose of calculating the statistical distributions.

For a categorical path A_i, we are interested in the possible values that the path might have on different objects. Let the domain of A_i be $1, 2, ..., d_k$. Then the categorical path domain is $D = \Pi_{i=1}^{k}$, which forms a contingency table. The probability of a particular categorical path can be approximated as $\widehat{\pi}_d = \frac{x_d}{n}$ where x_d is the frequency from the contingency table and $n = \sum_{d=1}^{D} x_d$. The path hash, frequency, and value of categorical data remain in the same format as the *value_data* file, and are output as the *categorical_values* component of the synthesis set.

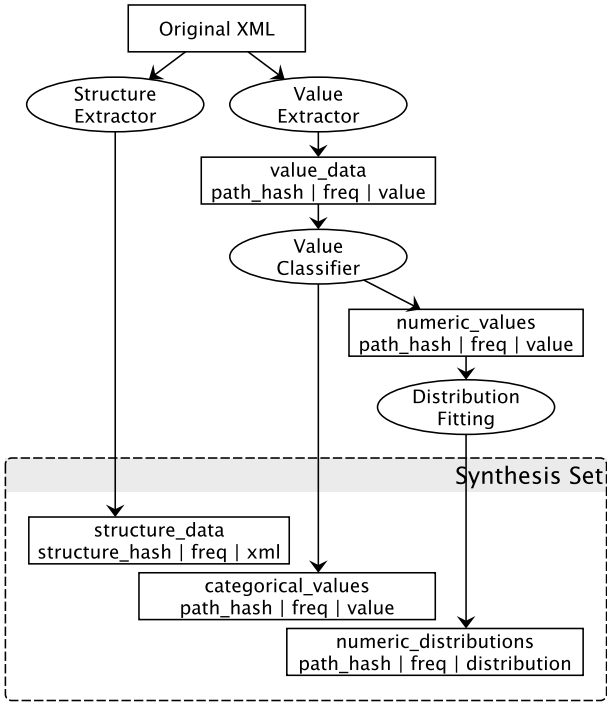

Fig. 1: Structure/Value Extraction – Cascading series of steps to extract the data patterns from complex XML documents.

Given a numeric path Z_j, we use a fitting method to determine the best distribution to represent the data, and then calculate n, max, and min. We consider a range of distributions that includes beta, Cauchy, chi-squared, exponential, F, gamma, geometric, log-normal, negative binomial, normal, Poisson, t, and Weibull distributions. JDistlib [14] is used to randomly generate numbers for the synthetic data from the maximum-likelihood fittings of the distributions.

IV. Synthetic Data Generation

The second major phase of the IoT synthetic data generator framework is the use of the synthesis set to create synthetic

XML data. The data generator has a few design requirements. First, it must scale to handle millions of structural representations, millions of unique paths, and many millions of possible categorical values. Secondly, it must support an effective method of anonymizing categorical values to provide an additional level of privacy for industrial and consumer information. In our approach, we use a simple string substitution mechanism to map every unique string in the *structure_data* and *categorical_values* components of the synthesis set to a randomly generated string of the same length as the original. A new obfuscated synthesis set is then generated using the string map which retains the qualities of correlation between tag names, attribute names, attribute values, and categorical values. The string map can be retained by the owner of the data for mapping synthetic results back to original strings. Note that the string substitution step is optional and could be replaced with a more sophisticated technique where one is needed. The synthetic generator performs identically with an original or anonymized synthesis set.

One design challenge is to manage millions of possible categorical values associated with a single path. In our test data some paths have over 5 million possible text values, each with an associated frequency of occurrence. In order to randomly select from these sets, we insert the possible values into unbalanced Huffman trees, implementing the linear time Huffman construction algorithm described in [15]. This allows for fast random selection, but it requires that the tree be kept in memory to avoid being rebuilt for each matching path.

Hadoop MapReduce was used to implement a distributed synthetic generator. One design goal in this case was to minimize the movement of the large amounts of categorical data across the network. Since any single structure could require loading many of the large value/frequency trees in the dataset, a design parameter of our approach was to group data that needed to be computed together. This translated to a design that performs two joins: one to group the structure hashes with a list of related path-value combinations, and another to join the path-value combinations to the structure. This is accomplished with a technique known as a reduce-side join (or repartition join), which we implemented in a manner similar to the Improved Repartition Join described in [16].

In the first phase of the two-part process, shown in Figure 2, the *structure_data*, *numeric_distributions*, and *categorical_values* are ingested by instances of the same map class. For structures, hashes for each path are computed as shown in Table II, and emitted with the frequency of occurrence and the structure hash in which they were discovered. Numeric and categorical values are passed through to the reduce phase. The interphase sort ensures that map outputs are grouped by their path hashes. Each reduce instance can then construct a single Huffman tree for the categorical values related to each of its assigned path hashes, and randomly select enough values to satisfy every structure. The generated values are keyed by the requesting structure hash, and emitted with the related path hash.

The second phase of the process begins with an identity mapper that ingests and emits the results of the first phase, keyed by structure hash, along with the structure data. The interphase sort groups all values by the structure hash. Finally, each reduce instance has a set of structures paired with enough values to fill them, accomplished by recursively generating path hashes to match to the input values as described above.

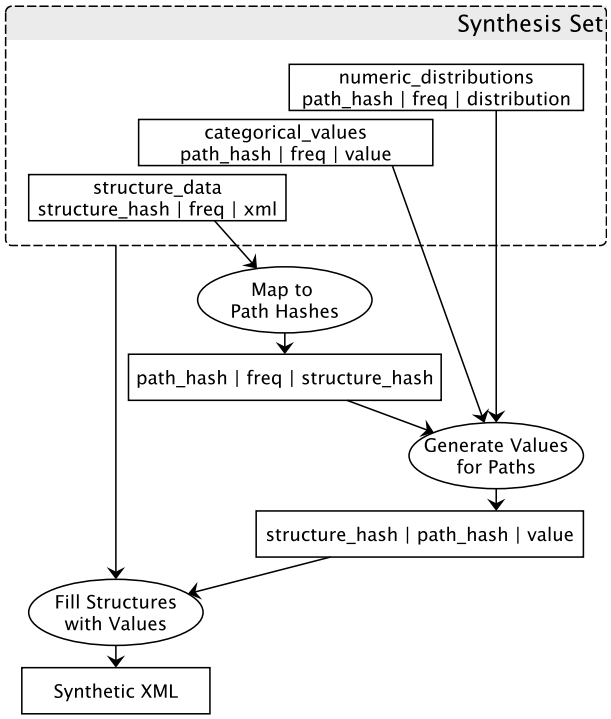

Fig. 2: XML Synthesis – The workflow of generating synthetic IoT data. Reduce-side joins are used to combine structure path hashes with possible values, the results of which are recombined with the structures to fill empty tags.

V. Evaluation

Our primary goal in generating synthetic data is to create an experimental framework that can be used to evaluate the performance tradeoffs of various data infrastructure tools for very complex data. Our framework creates synthetic data that have values and a structure that match the statistical characteristics of the original data. To evaluate our framework we start with an original data set that was collected from various sensors on 3,193,783 electronic objects over a period of several months. The total size of the data set is more than 3 terabytes. The data entries arrive to the data warehouse in XML format containing 6,347,462 individual XML documents. The dataset is preprocessed using a simple string substitution for anonymization of identifiers.

The framework is used successfully to extract descriptive statistics of the original data. Table III shows the descriptive statistics of the unique XML tags, attributes, and paths of these structures. While there are a fixed number of XML tags and XML attributes, there is no pre-determined XML schema or DTD document. Without a fixed schema, out of 6,347,462

XML structures, 5,758,590 are unique. The combination of different XML tags and attributes generate a total of 8,716,624 unique XML paths within the original data. Because XML tags, attributes, and paths can appear multiple times in a single XML structure, the tag seen most often appears more than 140 million times across the dataset and the most frequently seen attribute occurs more than 150 million times.

After extracting the unique XML structures and paths and identifying categorical and numeric values, the statistical distribution of each unique numeric value is also calculated. Just three distributions provide the best fit for 93% of the numeric paths: Poisson (58.1%), normal (23.5%), and geometric (11.7%). The geometric distribution used is $Pr(Y = k) = (1-p)^k p$, which models the number of failures before the first success. This level of statistical accuracy of the values meets our primary goals of synthetic data generation. The parameters of these distributions are recorded for use in the data generation phase of the framework. Development of techniques that provide high statistical accuracy of synthetic data values to original data is possible within the framework.

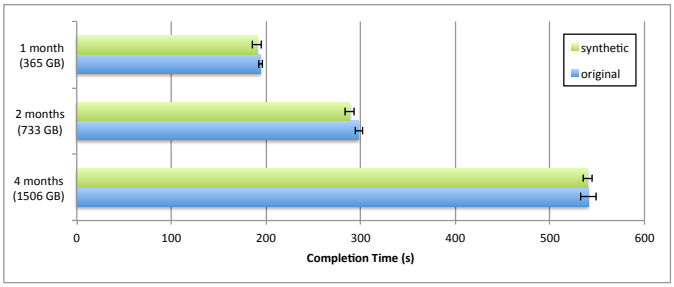

Fig. 3: Structural validation by performance comparison of tag-attribute searching, with 95% confidence interval.

We further evaluate the framework by comparing the performance of applications that use the IoT data when presented with the original and generated synthetic data of the same size. We use this as an additional measure of the structural similarity between the datasets. We generate data of equal size to the original IoT data collected in 1, 2, and 4 months, with sizes of 365, 733, and 1,506 GB, respectively. With each of these datasets we run a program to descend into each XML document and record the instances of certain XML tags with a specific attribute. We verify that the frequency of occurrence is as expected in the synthetic data, and use the runtime as a measure of the document complexity. Each test was run to a 95% confidence interval, as shown in Figure 3. In each of the test cases, the performance using synthetic data is similar to using the original data, supporting our assertion that the synthetic data is structurally similar to the original.

The final aspect of the evaluation of the framework is measurement of the time to generate synthetic data of various sizes from the descriptive statistics. To evaluate the distributed framework we create isolated testing environments for the generation process using the Clemson high performance computing cluster. In this environment we dynamically allocate four different Hadoop clusters with 10, 20, 30, and 40 compute nodes. Each node is configured with two 8-core Intel Xeon E5-2655 CPUs, 64 GB RAM, 1 Gb/s network link, and a single 900 GB HDD. The memory requirements of the reduce phase make it necessary to allocate 4 GB of RAM to each reducer JVM. To fit within memory requirements, we configure Hadoop to spawn 32 map tasks and 4 reduce tasks on each node. Synthetic data is generated in 1, 2, 4, and 8 month chunks, with sizes as described in Table IV.

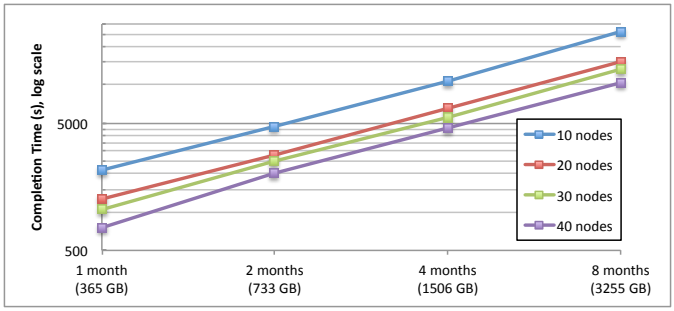

Fig. 4: Synthetic generation Phase 1.

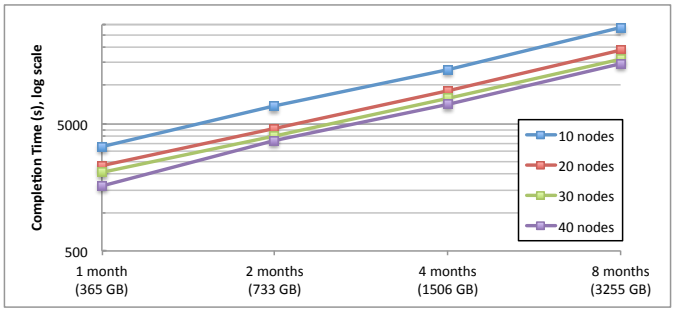

Fig. 5: Synthetic generation Phase 2.

In both phases of the data generation, we observe roughly linear scaling of the completion time as the number of compute nodes and size of the data vary, as shown in Figures 4 and 5. We note that the speedup from doubling the number of nodes is approximately 1.5x in Phase 1 and 1.3x in Phase 2. We hypothesize that this is due to the 1Gb/s network link between the nodes, which throttles the shuffle phase of map-reduce.

VI. Conclusions and Future Work

In this paper we present the design and implementation of a Hadoop-based synthetic IoT data generation framework. The main objective of this framework is to support research of infrastructure and tools for proprietary IoT data sets. The synthetic data generator enables the creation of large volume datasets that retain important characteristics of the data. The framework enables access by researchers to IoT data for the development and testing of tools and algorithms, and enables research by organizations that need to ensure the privacy of their sensitive data.

The modular design of the framework allows for streamlined extensions of the generator to support inclusion of different statistical distributions as necessary. In addition, the condensed

TABLE III: Descriptive statistics of the XML tags, attributes, and paths of the test data

	Individual XML Tags	Individual XML Attributes	Individual XML Paths	Individual Objects
Unique Count	110	29	8,716,624	3,193,783
Max Frequency	143,935,646	150,997,555	115,328,502	2,014
Min Frequency	11	1,383	1	1
Avg Frequency	1,966,444	75,34,407	2,582.4	1.99
Median	60,640	353,358	7.0	1.0

TABLE IV: Shuffle and output characteristics of the 2-phase MapReduce synthetic generator.

Input		Phase 1		Phase 2	
Months	Documents	Shuffle (gzip)	Output	Shuffle (gzip)	Output
1	1.61 million	371 GB (73 GB)	198 GB	487 GB (69 GB)	365 GB
2	3.09 million	715 GB (141 GB)	394 GB	975 GB (143 GB)	733 GB
4	6.35 million	1,470 GB (290 GB)	810 GB	2,004 GB (294 GB)	1,506 GB
8	13.72 million	3,177 GB (627 GB)	1,750 GB	4,331 GB (635 GB)	3,255 GB

form of structural and categorical data offers an option for ensuring privacy of confidential data within the synthetic data set. The preliminary efforts with string substitution have been successful. The use of Apache Hadoop and MapReduce for parallel processing makes the framework highly scalable, and enables a faster turnaround time in research and development.

Future work on the framework is possible. While the majority of the XML structures are unique in our test case, we need to support the case where there exists a common subtree in all of the XML structures. In this situation, pruning of the common subtree will help to reduce the size of the structural table and improve performance. The framework will be further modularized to allow users to specify their own set of distribution functions, including complex functions such as inter-dependent and correlated marginal distributions.

ACKNOWLEDGMENT

We would like to thank BMW for supporting this research. This work has been supported in part by NSF grant #1212680.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer networks*, vol. 54, 2010.

[2] McKinsey Global Institute, "Big data: The next frontier for innovation, competition, and productivity," 2011.

[3] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, "From the Internet of Things to the Web of Things: Resource-oriented architecture and best practices," in *Architecting the Internet of Things*. Springer, 2011, pp. 97–129.

[4] C. Aggarwal, N. Ashish, and A. Sheth, "The Internet of Things: A survey from the data-centric perspective," in *Managing and mining sensor data*. Springer, 2013, pp. 383–428.

[5] J. Cooper, A. James *et al.*, "Challenges for database management in the Internet of Things," *IETE Technical Review*, vol. 26, no. 5, 2009.

[6] R. Buyya, S. K. Garg, and R. N. Calheiros, "SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions," in *Procs. of the Int. Conf. on Cloud and Service Computing*, 2011.

[7] "TPC Benchmarks," 2014, available at http://www.tpc.org/information/benchmarks.asp.

[8] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen, "Bigbench: Towards an industry standard benchmark for big data analytics," in *Procs. of the ACM SIGMOD*, 2013.

[9] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Procs. of the ACM SoCC*, 2010.

[10] A. Aboulnaga, J. F. Naughton, and C. Zhang, "Generating synthetic complex-structured XML data." in *Procs. of WebDB*, 2001.

[11] S. Cohen, "Generating XML structure using examples and constraints," in *Procs. of the VLDB Endowment*, 2008.

[12] E. Maler, J. Paoli, C. M. Sperberg-McQueen, F. Yergeau, and T. Bray, "Extensible markup language (XML) 1.0 (third edition)," W3C, first Edition of a Recommendation, 2004, http://www.w3.org/TR/2004/REC-xml-20040204.

[13] T. White, *Hadoop: The Definitive Guide*. O'Reilly, 2012.

[14] "JDistLib," 2014, http://jdistlib.sourceforge.net.

[15] J. V. Leeuwen, "On the construction of Huffman trees," in *Proc. 3rd International Conference on Automata, Languages, and Programming*, Edinburgh University, 1976.

[16] S. Blanas, J. Patel, V. Ercegovac, J. Rao, E. Shekita, and Y. Tian, "A comparison of join algorithms for log processing in MapReduce," in *Procs. of ACM SIGMOD*, 2010.