Domain-Driven, Generative Data Model for BigPetStore

Ronald J. Nowling
Red Hat, Inc.
Raleigh, NC 27601
Computer Science & Engineering,
University of Notre Dame,
Notre Dame, IN 46617
Email: rnowling@redhat.com

Jay Vyas
Red Hat, Inc.
Raleigh, NC 27601
Email: jvyas@redhat.com

***Abstract*—Access to high-quality, semantically-rich data sets and complex, realistic analytics workflow examples that can scale from local development environments to large clusters are key for consistent development, testing, and benchmarking of big data software, algorithms, and infrastructure. BigPetStore is a big data application blueprint, which incorporates a data generator and complex examples of cleaning, aggregation, and analytics, built around a fictional chain of pet store which has been used for testing and benchmarking in the Apache BigTop distribution. We present a new generative data model for BigPetStore and demonstrate the model's ability to generate semantically-rich data that can be used for complex analytics workflows at variable scale ranging from a single machine to a large cluster.**

***Keywords*—big data, synthetic data sets, data generation, benchmarking, testing, probabilistic models**

I. Introduction

Big data applications process large, continuously-changing data sets with the general goal of information and knowledge extraction. New tools such as Hive [1], Pig [2], [3], and Shark [4] based around big data frameworks such as Hadoop [5], Storm [6], and Spark [7], [8] are being released every day. Development, testing and benchmarking of big data software, algorithms, and infrastructure pose unique challenges. Though software and algorithms will be deployed to large clusters for benchmarking and production usage, engineers develop and test these tools on desktops due to the speed and ease of development and debugging on a local machine and limited access to large clusters due to the high costs of purchasing, provisioning, and maintaining such clusters. Software and algorithms are being developed at incredible speed with new releases everyday. The high speed of development driven by the need to be the first to market in a competitive field often results in buggy software. There is great need for consistent and rigorous testing and benchmarking frameworks to aid in the identification of bugs while maintaining the high-speed of development. Access to high-quality, semantically-rich data sets and complex, realistic analytics workflow examples that can scale from local development environments to large clusters are key to increasing access to and development of consistent and rigorous testing and benchmarking frameworks.

Over time, more high-quality data sets are becoming publicly available, but these data sets often carry burdens that force developers to make undesirable trade offs. Since few data sets are scalable (allowing the user to pick how much data they want), users are forced to choose between small data sets which are useful for development and testing on a local machine, but not for production-scale tests or benchmarking, or large data sets which are prohibitively expensive to host, transfer, and store. Furthermore, access to large data sets can be unreliable due to dependence on third party benefactors for hosting and requires significant amounts of bandwidth to download. Several companies (such as Twitter [9]) offer access to their data but the costs and usage restrictions can be undesirable.

Methods for generating synthetic data offer multiple advantages over pre-generated data sets. The code for generating the data is often small, making it easy and cheap to host, transfer, and store. Licensing restrictions and data privacy issues are avoided with synthetic data sets. And, the size of the data sets can be scaled arbitrarily, enabling seamless usage of the same data set across a range of situations.

Several synthetic data set generators exist. TeraGen and the Intel Hadoop Benchmark Suite [10] can generate data sets quickly and at any scale. Targeted towards specific benchmarks such as Map-Reduce jobs and Hive queries, the resulting data is semantically-void and not useful for other tasks. Multiple frameworks exist for generating synthetic data sets that satisfy relational database schemas [11]–[17]. Several approaches use domain-specific languages [16], [17] to allow users to specify additional constraints such as which distributions to sample from or allow modeling basic relationships between fields such as simple arithmetic equations [18]. Arasu, et al. [19] demonstrated that constraint-solving techniques could be used as an alternative to procedural approaches. These frameworks allow for reproducing the structural properties of real data, but the frameworks are not expressive enough to describe the dynamic generation processes and latent variables that would be needed to embed the desired informational complexity and rich semantics needed for analytics.

Realizing the difficulty in creating a generic framework capable of modeling the semantics of deep domain knowledge, recent work [20] has focused on generating synthetic data sets that satisfy characteristics learned from a real reference data sets. Such approaches appear promising but will need to overcome the difficulties of accurately training models, especially on raw data instead of features.

Until such methods have reached maturity, the development of customized, domain-driven models is best suited for satisfying the needs of big data software and algorithms development. Several customized, domain-driven models have seen usage in real-world situations. Information Discovery and Analysis System (IDAS) Data Set Generator (IDSG) [21], [22] was developed for the Department of Homeland Security to develop synthetic data for effectiveness of IDASs designed to identify potential terrorism threat scenarios.

Huppler, Pavlo. standard benchmarks

The BigPetStore application aims to be a big data application blueprint built around a fictional chain of pet stores, which incorporates a data generator and complex examples of cleaning, aggregation, and analytics using well-known tools such as Hive and Mahout [23] from across the Hadoop and Spark ecosystems. BigPetStore's usefulness has been demonstrated by its incorporation into the open-source Apache BigTop distribution [24] for testing and as a reference application.

In this work, we present a new domain-driven model and simulation for generating arbitrarily large, complex data for testing and benchmarking of algorithms, software, and infrastructure in the big data space. Compared with BigPetStore's previous data generator, our model and implementation can generate data which contains geospatial, temporal, and quantitative features, similar to the type of data which businesses and organizations might typically encounter. Combining the features of the TeraGen (scalability) and MovieLens [25] (semantically rich) input data sets commonly used for big data benchmarking, we present a scalable synthetic data set generation implementation which can be used to benchmark lower level tasks (such as sorting) as well as higher level tasks (such as clustering, recommending, and search indexing) at arbitrary scale. To validate this approach, we perform an example analysis designed to guide a fictional advertising campaign (a typical business use case) to demonstrate that the model generates data with desirable properties (global patterns and region specific transactions). We have made the model's implementation available as open source software.

II. Overview of Model & Simulation Procedures

Our generative model combines various well-known mathematical modeling techniques to simulate the factors affecting customers' purchasing habits. Each part of the model is derived from *ab initio* assumptions. In several cases, real data is used to parameterize parts of the model.

A. Relational Data Model

The model generates data for four types of records: stores, customers, transactions, and transaction items (Figure 1). Store records contain both a unique identifier and a location in the form of a 5-digit zip code. Customer records consist of a unique identifier, a name, and a location given as a 5-digit zip code. Transaction records consist of a customer identifier, a store identifier, and a time given in days since the beginning of the simulation. Transaction item records contain a transaction identifier and an item description. The item description is a list

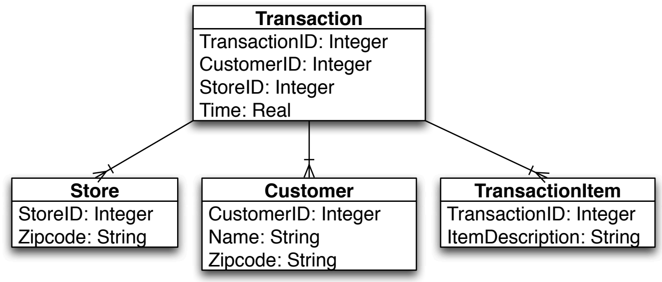

Fig. 1. Relational Data Model for Generated Data

of key-value pairs stored as a string. Key-value pairs are used since item characteristics differ depending on the type of item. For example, dog food has a brand, a flavor, a size (in lbs), and a per-unit cost, while poop bags have a brand, a color, a count, and a per-unit cost.

B. Generation of Stores

The locations of the stores are modeled by a probability density function (PDF) that gives the probability that a store's location is the given zip code. We designed the PDF to give to zip codes in high-population, high-income areas the highest probabilities. The PDF is composed of two individual PDFs. One PDF determines the probability of each zip code as the population of that zip code over the total population:

$$p(\text{location} = z|\text{population}(z)) = \frac{\text{population}(z)}{\sum_i \text{population}(i)}$$

The second PDF scales the probabilities of the zip codes by their incomes. The zip code with the highest-income has a probability s-times larger than that of the lowest-income zip code. The values in-between are interpolated using an exponential function:

$$p(\text{location} = z|\text{income}(z)) = s^{\left(\frac{\text{income} - \min_i \text{income}(i)}{\max_i \text{income}(i) - \min_i \text{income}(i)} - 1\right)}$$

The combined PDF is given as:

$$\begin{aligned} p(\text{location} = z) =& Z^{-}1 p(\text{location} = z|\text{population}(z)) \\ & p(\text{location} = z|\text{income}(z)) \end{aligned}$$

where Z is the normalization factor. Given the small size (≈30,000 zip codes) of the data set, Z can be found directly by iterating over all of the zip codes and summing the scores. The population and household income data for the zip codes were taken from from the U.S. Census American Community Survey [26]. The stores' locations are generated by sampling zip codes from the PDF.

C. Generation of Customers

For each proposed customer location zip code z, we compute the distance $d_m(z)$ between z and each store's zip code s_z to find the closest store. The zip codes' latitudes and longitudes (taken from the Zip Code Database Project [27]) are used to compute the distances. Each zip code z is assigned a weight w_z according to its distance $d_m(z)$ to the nearest store using an exponential distribution with the average distance β:

$$\begin{aligned} w_z &= \beta^{-1} \exp(-\beta^{-1} \, d_m(z)) \\ d_m(z) &= \min_{s_z} \, d(z, s_z) \end{aligned}$$

The customer's zip code is chosen by sampling from the zip codes with the probabiltiy of each zip code z proportionate to its weight w_z.

Names are generated using data from the Name Database [28]. Each record in the database gives a name, a weight, and flag indicatings if the name can be used as a first name, a last name, or both. PDFs generated for the first and last names using the weights. The customer's name is generated by sampling a first name and a last name from each PDF respectively.

We determine the number of pets N_p each customer has by sampling from a discrete uniform distribution of integers. We then sample the number of dogs N_d as a discrete uniform distribution of integers between 0 and N_p. The remaining number of pets are assigned to be cats.

$$\begin{aligned} N_p &\sim U(1, b) \\ N_d &\sim U(0, N_p) \\ N_c &= N_p - N_d \end{aligned}$$

D. Simulation of Transactions

A transaction simulation is run for each customer. The transaction's store is set to the store located closest to the customer. Transaction times are generated from a Monte Carlo process that proposes transaction times based on the usage of the customer's items and Poisson processes modeling the amount of time between the customer visiting the store and running out of the items (Section II-D1). The purchased items are generated by a Hidden Markov Model (HMM) parameterized by the transaction time and amount of time remaining for the items in the customer's inventory (Section II-D2).

1) Simulation of Transaction Times: Transaction times are simulated using a Monte Carlo method (Figure 2). The usage over time of each item category is simulated. Items categories are groups of items which are interchangeable such as "dry dog food," "dry cat food," and "kitty litter." The time between the customer's visit to the store to buy more items in each category and the exhaustion time of that item category is modeled using a Poisson process. Proposed transaction times for each item category are computed based on the exhaustion time and time sampled from the Poisson process. The earliest transaction time is taken as the overall proposed transaction time. The probability of the transaction time is calculated using a PDF.

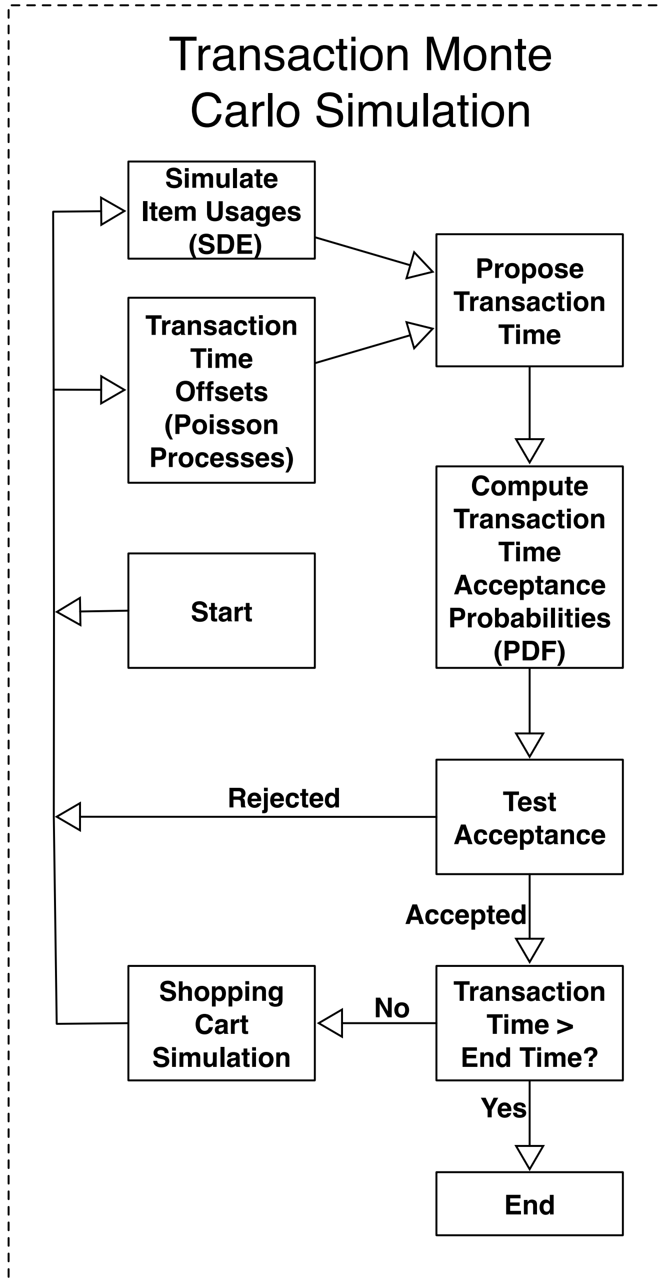

Fig. 2. Flowchart of Transaction Monte Carlo Simulation

If rejected, new transaction times are proposed. Otherwise, the purchased items are modeled using a separate process (discussed below). After the items are chosen, the process begins again with the simulation of the item category usages.

In the first stage of the process, the usage of items over time is modeled using a stochastic differential equation (SDE):

$$\frac{da_i}{dt} = -\mu_i + \sigma_i \, dW_i(t)$$

where a_i is the amount of item category i "stuff" remaining at time t. The variable u_i gives the average usage rate and σ_i^2 gives the variance of the usage rate for item category i.

The SDE is numerically integrated using the Euler-Maruyama method [29] with time step Δt_n sampled from an exponential distribution:

$$\begin{aligned}
&a_{n+1,i} = a_{n,i} - \mu_{i,c}\,\Delta t_n + \sigma_{i,c}\sqrt{\Delta t_n} R_n \\
&\Delta t_n \sim \mathrm{Exp}(\lambda_i) \\
&R_n \sim \mathrm{N}(0,1)
\end{aligned}$$

where λ_i is the item category's average usage rate, μ_i is the item category's average amount used per time used, and σ_i^2 is the variance of the amount of the item category used per time used. The parameter λ_i is defined for each item category. The parameters $\mu_{i,c}$, and $\sigma_{i,c}^2$ are found based on a base rate for each item category multiplied by the number of pets of the appropriate species the customer has. The exhaustion time $T_{E,i}$ for each item category is found by simulating the usage until $a_n = 0$.

The proposed transaction time is found from the exhaustion times (Eq. 1). For each item category, the offset time $T_{O,i}$ between when a customer would visit the store and the exhaustion time is sampled from an exponential distribution parameterized separately for each customer c. A proposed transaction time $T_{P,i}$ for each item category is found by subtracting the offset time $T_{O,i}$ from the exhaustion time $T_{E,i}$. The earliest proposed transaction time is taken as the overall proposed transaction time T_T.

$$\begin{aligned}
&\beta_c \sim \mathrm{U}(a,b) \\
&T_{O,i} \sim \mathrm{Exp}(\beta_c^{-1}) \\
&T_{P,i} = T_{E,i} - T_{O,i} \\
&T_T = \min_i \{T_{P,i}\}
\end{aligned} \tag{1}$$

The acceptance probability $p(t_{n+1} = T_T|t_n)$ of the proposed transaction time T_T is modeled using a PDF (Eq. 2). For now, the PDF only ensures that the proposed transaction time is more recent than the last transaction time.

$$p(t_{n+1} = T_T|t_n) = \begin{cases} 1 & t_{n+1} \geq t_n \\ 0 & t_{n+1} < t_n \end{cases} \tag{2}$$

The proposed transaction time is accepted if $p(t_{n+1} = T_T) < r$, where $r \sim \mathrm{U}(0,1)$. Until the proposed transaction time is accepted, a new proposed transaction time is generated by sampling new offset times. If the proposed transaction time is accepted, the shopping cart of items is generated through a separate simulation discussed in Section II-D2 below. Transactions are generated until the accepted transaction time is later than the end time given as a simulation parameter.

2) Simulation of Transaction Items: A customer's purchases in each transaction are modeled using a layered Hidden Markov Model (HMM) (Figure 3). The HMM has three types of states: the start state, the purchase states, and the stop state. There are an infinite number of purchase states, identified by the exhaustion times of the customer's item categories, the transaction time, and the number of items purchased in the current transaction. The observables of the states correspond to the item categories. Each item category i is assigned a weight w_i according to the amount of time between the transaction time T_t and $T_{E,i}$ when each item category will be exhausted (based on the usage simulations):

$$w_i = -\lambda_i \exp(-\lambda_i(T_{E,i} - T_T))$$

The item category for each purchase is chosen by sampling from the item categories with the probability of each item category i proportionate to its weight w_i.

Markov Models are used to model the customer's purchasing behavior for each item category. Each state X_n corresponds to an item in that category. The customer's buying habits are determined by the transition probabilities (Eq. 3). The transition probabilities depend on three types of parameters: loopback weight w_l indicating the probability of purchasing the same item again, weight w_f of a particular field f, and the weight $w_{f,s}$ when two items x and y have the same value for the field f.

$$Pr(X_{n+1} = x|X_n) = \begin{cases} (1.0 - w_l)W^{-1}\sum_f w_f w_{f,s}(x,y) & \text{if } x \neq X_n \\ w_l & \text{if } x = X_n \end{cases} \tag{3}$$

todo

$$w_{f,s}(x,y) = \begin{cases} w_{f,s} & \text{if } x.f = y.f \\ 1 - w_{f,s} & \text{if } x.f \neq y.f \end{cases}$$

todo

$$W = \sum_i \sum_{j \neq i} \sum_f w_f w_{f,s}(i,j)$$

The weights w_l, w_f, and $w_{f,s}$ are chosen randomly for each customer.

description. min, max

$$\begin{aligned}
&w_l \sim N(\mu, \sigma^2) \\
&w_f \sim N(\mu, \sigma^2) \\
&w_{f,s} \sim N(\mu, \sigma^2)
\end{aligned} \tag{4}$$

reorder so that probilities describe habits

For example, if a customer tends to buy the same brand of dog food repeatedly, the outgoing edges to all products of the same brand as the product in the current state will be higher than those for other brands' products. If the customer does not care about flavor, the outgoing edges to all products from the current brand will be equal, otherwise the edge weights can be adjusted to prefer a specific combination of brand and flavor. Likewise, if a customer cares more about flavor or cost, the edge weights will be assigned appropriately.

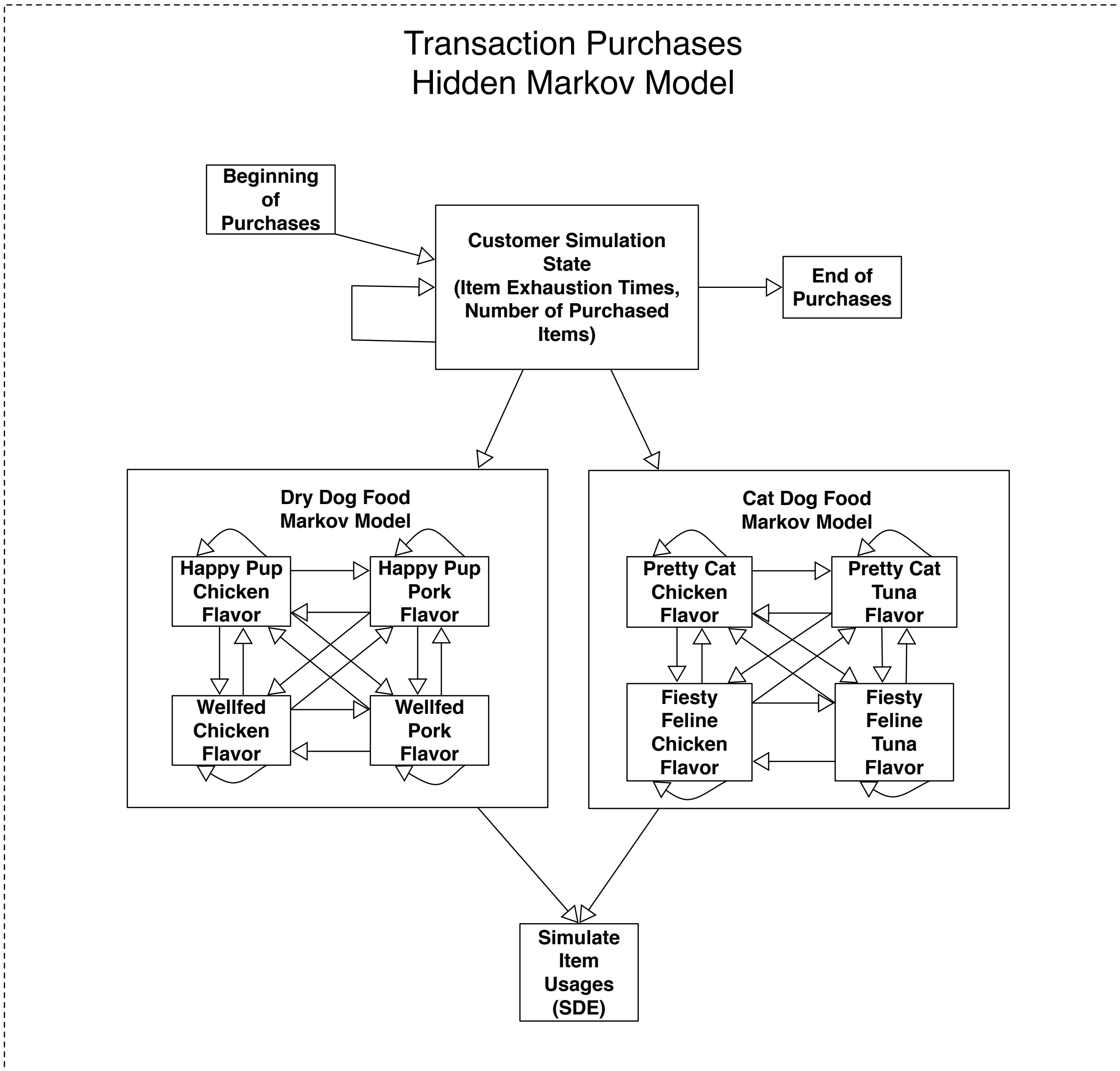

Fig. 3. Example Shopping Cart Hidden Markov Model

To simulate an item purchase, the Markov model is transitioned forward by one state. The current states of the Markov models are kept between purchases.

After an item is chosen, the item category's exhaustion time is updated by propagating the item category usage SDE described in Section II-D1. state transition, stop state

The HMM can be extended to consider additional factors such as the amount of money spent so far in the transaction and the customer's spending habits.

III. Implementation

The models and simulations were implemented using Python. The source code is available on GitHub [30] at https://github.com/rnowling/bigpetstore-data-generator under the Apache 2 license.

IV. Evaluation of the Model

In accordance with our original design goals, we evaluated the model and implementation in terms of their abilities to generate semantically-interesting data and scale from a single local machine to a cluster.

A. Example Analysis of Generated Data

To evaluate the semantic content of the generated data, we performed two example analyses in support of a fictional advertising campaign using data for 10 stores, 10,000 customers, and five years of transactions generated from the model. The analyses were designed to be realistic and driven by real-world business concerns. Due to limited advertising budgets, businesses need to decide who to target and what sort of advertisements are likely to be effective for each type of customer. We analyzed data generated from the model to determine where the advertising campaigns should be targeted geographically and identify customer purchasing profiles.

First, we wanted to identify how close most customers live to a store since advertising to people who live or work too far away from a store is unlikely to be effective and will increase our costs. We analyzed the distribution of distances between customers' homes and their nearest stores. We found that most

Fig. 4. Distributions of Distances of Customers to Closest Stores

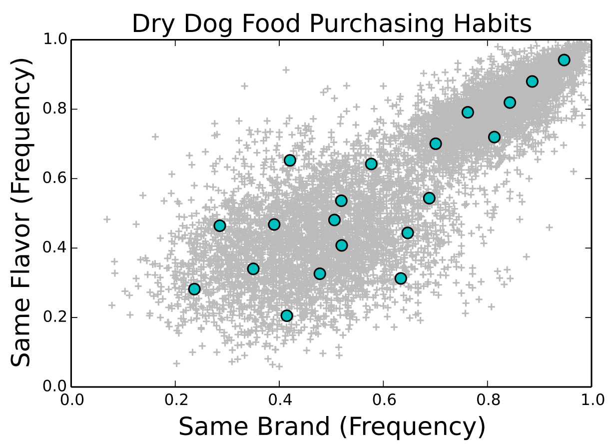

Fig. 5. Clustering of Brand and Flavor Purchasing Preferences. Grey plus signs (+) represent customer data points, and cyan circles represent cluster centers.

customers tend to live within 15 miles of their closest store, suggesting that we should limit our advertising to a 15-mile radius around each our store.

Secondly, we profiled our customers' purchasing habits to optimize the advertising campaign's effectiveness by customizing the advertised products for each customer. For each customer, we generated a feature vector by computing the frequency with which they would purchase the same brand or flavor from one transaction to the next. We clustered the customers' feature vectors using the KMeans algorithm with a range of cluster counts. Twenty clusters converged the error.

Analysis of the feature vectors and clusters revealed four predominant purchasing profiles (Figure 5). High frequencies of purchasing the same flavors repeatedly were tightly-correlated with high frequencies of purchasing the same brands – these customers were likely to be happy with a particular item and kept purchasing the same item repeatedly. For our advertising campaign, it would be unlikely to get these customers to purchase different items, so we should create incentives to purchase larger quantities, especially when inventory levels are high and needed to be depleted. Other customers had a tendency to purchase either the same flavor or brand repeatedly,

TABLE I. BENCHMARKS OF DATA GENERATOR

Stores	Customers	Simulated Time (years)	Transactions	Data Size (MB)	Run Time (min)
10	10,000	1	279,870	18	3.5
100	10,000	1	279,586	18	3.5
10	1,000	5	123,309	8	1.1
100	1,000	5	127,064	8	1.2
10	10,000	5	1,275,542	84	11.0
100	10,000	5	1,268,403	84	10.5

but varied in their choice of the other. For customers who prefer a particular brand, we should target our advertising campaign to suggest other flavors sold by that brand. Likewise, for customers who prefer a particular flavor, we should target our advertising campaign to suggest other brands with that flavor. Lastly, some customers had very low frequencies of purchasing neither the same brand nor flavor. Other factors, such as cost, not represented in this analysis may be driving these customers' purchasing habits and will need further study.

B. *Scaling of Data Size and Run Time*

BigPetStore aims to scale from a local desktop to a large cluster. To evaluate the scaling of the model and implementation, we benchmarked the data generator on a laptop with a 2 GHz Intel Core i7 CPU, 8 GB of RAM, and a 256 GB SSD using the Python implementation with a single thread. Using the test setup, between 1,500 and 2,000 transactions can be generated per second (Table I). As the customers' transaction simulations are independent of one another, the transaction generation can easily be parallelized so that 1,500-2,000 transactions can be generated per thread per second.

The number of transactions (and hence, data size) grows as $O(N_c T)$ where N_c is the number of customers and T is the amount of time to be simulated. The amount of data generated can be scaled as large as necessary simply by increasing N_c and T. Five years of transactions for 100,000 customers will generate approximately 1 GB of data. A terabyte of data can be generated by simulating 100 million customers over five years.

V. DISCUSSION AND CONCLUSION

We have described a domain-driven mathematical model and accompanying simulation for generating semantically-rich data. We validated the method by analyzing generated data to inform decisions in a fictional advertising campaign. Scalability testing combined with analysis of the implementation suggests that the data generation method can scale from small data sets appropriate for desktop development to large data sets for testing and benchmarking of clusters. We have released the implementation source code under the Apache 2 open-source license.

We see many opportunities for building on the work presented. We intend to expand the model to incorporate additional factors. Deeper integration of existing fields such as location and purchasing profiles (e.g., to model regional purchasing preferences) will increase the variety of the semantic information encoded in the generated data. The incorporation of weather and climate data can be used to influence when customers shop and the types of products they buy. For example, customers would be less likely to shop during snow storms and more likely to buy winter apparel for their pets. Modeling of time-dependent events such as sales, evolution of customer purchasing profiles over time, and "life events," such as the birth or passing of pets, will enable more interesting time-series analysis of the data. We would also like to expand the scope of the model to incorporate business processes (such as inventory management, customer complaints, employees, etc.), thus enabling queries about how internal business process affect customer behavior.

The current implementation was prototyped in Python. We are working to implement the model using Hadoop and Spark. We will commit the resulting implementations to the Apache BigTop project to enable ease of access and immediate benefit to current users.

ACKNOWLEDGMENT

The authors would like to thank Brian P. Clare and Casey Robinson for useful and interesting discussion. JV would like to thank the Apache BigTop community for their interest in and support of BigPetStore. We would also like to thank Matt Fenwick, Nigel Savage, and Bhashit Parikh for their contributions to BigPetStore. The authors would like to thank Red Hat, Inc. and the University of Notre Dame for their support of this work.

REFERENCES

[1] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive - a petabyte scale data warehouse using Hadoop," in *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. Ieee, 2010, pp. 996–1005.

[2] C. Olston, B. Reed, and U. Srivastava, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 1099–1110.

[3] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, "Building a high-level dataflow system on top of Map-Reduce: the Pig experience," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1414–1425, 2009.

[4] C. Engle, A. Lupher, R. Xin, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: fast data analysis using coarse-grained distributed memory," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012, pp. 689–692.

[5] "Apache hadoop," http://hadoop.apache.org/.

[6] "Apache storm," http://storm.incubator.apache.org/.

[7] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010, pp. 1–7.

[8] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012, pp. 1–14.

[9] "Twitter public streams," https://dev.twitter.com/docs/streaming-apis/streams/public.

[10] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*. IEEE, 2010, pp. 41–51.

[11] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolette, and H.-A. Jacobsen, "Bigbench: Towards an industry standard benchmark for big data analytics," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 1197–1208.

[12] T. Rabl and M. Poess, "Parallel Data Generation for Performance Analysis of Large, Complex RDBMS," in *Proceedings of the Fourth International Workshop on Testing Database Systems*, 2011, pp. 1–6.

[13] M. Frank, M. Poess, and T. Rabl, "Efficient update data generation for DBMS benchmarks," in *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering - ICPE '12*. Boston, Massachusetts, USA: ACM Press, 2012.

[14] T. Rabl, M. Frank, H. M. Sergieh, and H. Kosch, "A data generator for cloud-scale benchmarking," in *Performance Evaluation, Measurement and Characterization of Complex Systems*, R. Nambiar and M. Poess, Eds. Springer, 2011, pp. 41–56.

[15] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger, "Quickly generating billion-record synthetic databases," in *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, 1994, pp. 243–252.

[16] N. Bruno and S. Chaudhuri, "Flexible Database Generators," in *VLDB '05 Proceedings of the 31st international conference on Very large data bases*, 2005, pp. 1097–1107.

[17] J. E. Hoag and C. W. Thompson, "A parallel general-purpose synthetic data generator," *ACM SIGMOD Record*, vol. 36, no. 1, pp. 19–24, Mar. 2007.

[18] A. Alexandrov, K. Tzoumas, and V. Markl, "Myriad: scalable and expressive data generation," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1890–1893, 2012.

[19] A. Arasu, R. Kaushik, and J. Li, "Data generation using declarative constraints," in *Proceedings of the 2011 international conference on Management of data - SIGMOD '11*. New York, New York, USA: ACM Press, 2011, p. 685.

[20] A. Alexandrov, C. Brücke, and V. Markl, "Issues in big data testing and benchmarking," in *Proceedings of the Sixth International Workshop on Testing Database Systems - DBTest '13*. New York, New York, USA: ACM Press, 2013, p. 1.

[21] D. R. Jeske, B. Samadi, P. J. Lin, L. Ye, S. Cox, R. Xiao, T. Younglove, M. Ly, D. Holt, and R. Rich, "Generation of synthetic data sets for evaluating the accuracy of knowledge discovery systems," in *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining - KDD '05*. New York, New York, USA: ACM Press, 2005, p. 756.

[22] P. Lin, B. Samadi, and A. Cipolone, "Development of a synthetic data set generator for building and testing information discovery systems," in *Information Technology – New Generations, 2006*. Las Vegas, NV: IEEE, 2006, pp. 707–712.

[23] "Apache mahout," https://mahout.apache.org/.

[24] "Apache bigtop," http://bigtop.apache.org/.

[25] "Movielens," http://www.grouplens.org/datasets/movielens/.

[26] "American community survey," http://www.census.gov/acs/www/.

[27] "Zip code database project," http://zips.sourceforge.net/.

[28] "https://www.drupal.org/project/namedb," http://incompetech.com/named/.

[29] P. E. Klouden and E. Platen, *Numerical Solution of Stochastic Differential Equations*. Springer, 2013.

[30] "Github," https://github.com/.