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*—Both developers and users of big data systems benefit from realistic example applications. For developers, such as applications are useful for testing, benchmarking, and evaluating design choices; for users, example applications provide starting points for learning methods, developing their own applications, and implementing new analytics workflows.**

We present a new domain-driven, generative data model for BigPetStore, a big data application blueprint for the Hadoop ecosystem and included in the Apache BigTop distribution. We describe the model and demonstrate the its ability to generate semantically-rich data at variable scale ranging from a single machine to a large cluster. We validate the model by using the generated data to answer questions about customer locations and purchasing habits for a fictional targeted advertising campaign, a common business use case.

***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. With the wide variety of big data tools available and lagging documentation, both developers and users of big data systems benefit from realistic example applications. For developers, such as applications are useful for testing, benchmarking, and evaluating design choices; for users, example applications provide starting points for learning methods, developing their own applications, and implementing new analytics workflows.

BigPetStore is a big data application blueprint built around processing transaction data for a fictional chain of pet stores. BigPetStore mainly targets the Hadoop [1] ecosystem with examples implemented for loading, cleaning, aggregating and performing analytics on data using Hive [2], Pig [3], [4], and Mahout [5]. BigPetStore has been incorporated with the open-source Apache BigTop distribution [6], where it is used for testing and as a reference application. One of BigPetStore's key features is the ability to scale from a single machine to a large cluster, making it easy to develop projects on a local machine and transfer the application to a large cluster for production testing.

To achieve the project's goals of providing high-quality, realistic examples, BigPetStore requires semantically-rich, complex data. At its core, BigPetStore relies on a generative data model for producing synthetic transaction data. Despite the growing number of real data sets now publically available, synthetic data generators have a number of advantages for BigPetStore over real data sets. The synthetic data generator can be packaged with BigPetStore, avoiding the cost, time, and infrastructure needed to host, transfer, and store large data sets. Synthetic data generators are scalable, allowing the user to choose how much data to generate – a requirement for supporting BigPetStore's goal of running on both single machines and large clusters. Licensing and privacy issues are avoided with the use of synthetic data sets. And lastly, synthetic data generators are tunable by the user, allowing the user to generate data with specific criteria amenable to testing. For example, a user may generate transactions from a small number of purchasable items so that clustering results can be easily visualized.

A variety of approaches for generating synthetic data set exist. TeraGen and the Intel Hadoop Benchmark Suite [7] are popular tools that can generate data sets quickly and at any scale, but the resulting data is semantically-void and not useful for much more than simple benchmarks. Multiple frameworks exist for generating synthetic data sets that satisfy relational database schemas [8]–[14]. Several approaches even provide domain-specific languages [13], [14] for specifying additional constraints such as which distributions to sample from and allow for modeling basic relationships between fields using simple arithmetic equations [15]. Arasu, et al. [16] 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 BigPetStore's analytics examples.

Realizing the difficulty in creating a generic framework capable of modeling the semantics of real data, recent work [17] has focused on generating synthetic data sets that satisfy characteristics learned from real reference data. 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, BigPetStore's needs are best met by a customized, domain-driven model. Another example of such a model is The Information Discovery and Analysis System (IDAS) Data Set Generator (IDSG) [18], [19]. IDSG was developed to to test the effectiveness of IDASs in identifying potential terrorism threat scenarios using synthetic data. Like BigPetStore, the developers of IDSG needed to avoid licensing restrictions and privacy issues associated with real data.

BigPetStore's current model has been used successfully to generate terabytes of synthetic data and is used regularly for testing in Apache BigTop, showcasing the value of the approach. The current model is limited, however, in its ability to generate data that is semantically rich, limiting progress on BigPetStore's goals of providing realistic analytics examples.

In this work, we present a new domain-driven model and simulation method for BigPetStore. 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 [20] (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 demonstrate that the model generates data with desirable properties, we performed an example analysis designed to guide a fictional advertising campaign (a typical business use case). 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 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:

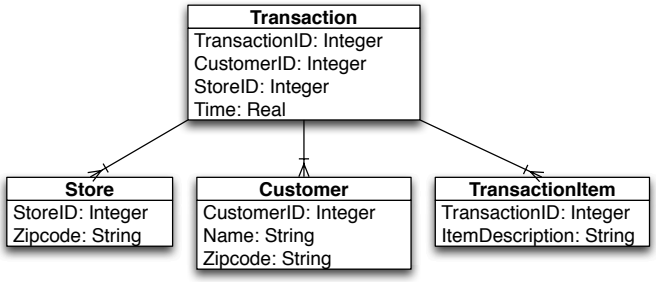

Fig. 1. Relational Data Model for Generated Data

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

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 ($\approx$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 [21]. 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 [22]) 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 [23]. 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.

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

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. 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):

$$\frac{da_i}{dt} = \min\{-\mu_i + \sigma_i \, dW_i(t), 0\}$$

where a_i is the amount of item category i "stuff" remaining at time t. The variable μ_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

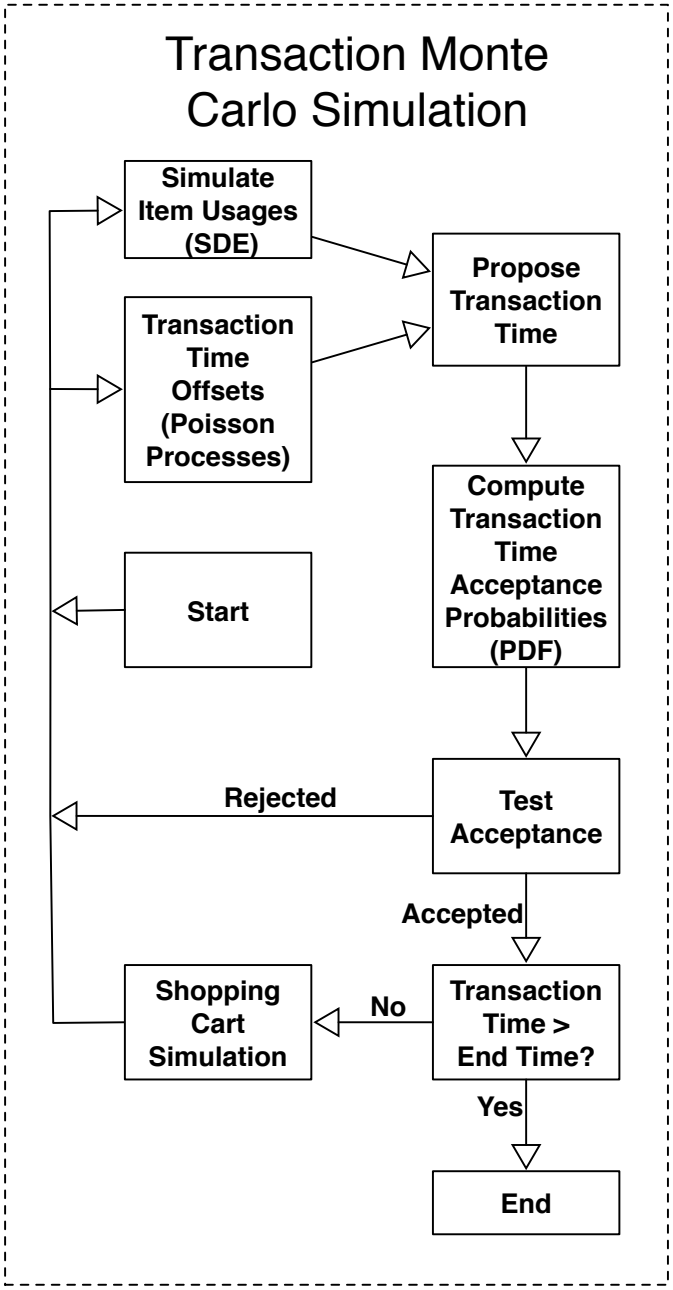

Fig. 2. Flowchart of Transaction Monte Carlo Simulation

method [24] with time step Δt_n sampled from an exponential distribution:

$$\begin{aligned} a_{n+1,i} &= a_{n,i} - \min\{\mu_{i,c} \, \Delta t_n + \sigma_{i,c} \sqrt{\Delta t_n} R_n, 0.0\} \\ t_{n+1} &= t_n + \Delta t_n \\ \Delta t_n &\sim \text{Exp}(\beta_{i,c}^{-1}) \\ R_n &\sim \text{N}(0, 1) \end{aligned}$$

where $a_{n,i}$ is the amount remaining of item category i at time step t_n, $\mu_{i,c}$ is the average amount used per time used, $\sigma_{i,c}^2$ is the variance of the amount used per time used, and $\beta_{i,c}$ is the item category's average amount of time between uses

for customer c. The parameters $\mu_{i,c}$, and $\sigma^2_{i,c}$ are computed from a base rate for each item category multiplied by the number of pets of the appropriate species the customer c has. The exhaustion time $T_{E,i}$ for each item category is found by simulating the usage until $a_{n,i} \leq 0$.

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. β_c

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

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 $r \sim \text{U}(0, 1)$. 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 purchased items are chosen 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. λ_i - maybe these should be per customer / item category instead of just per item category

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.

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.

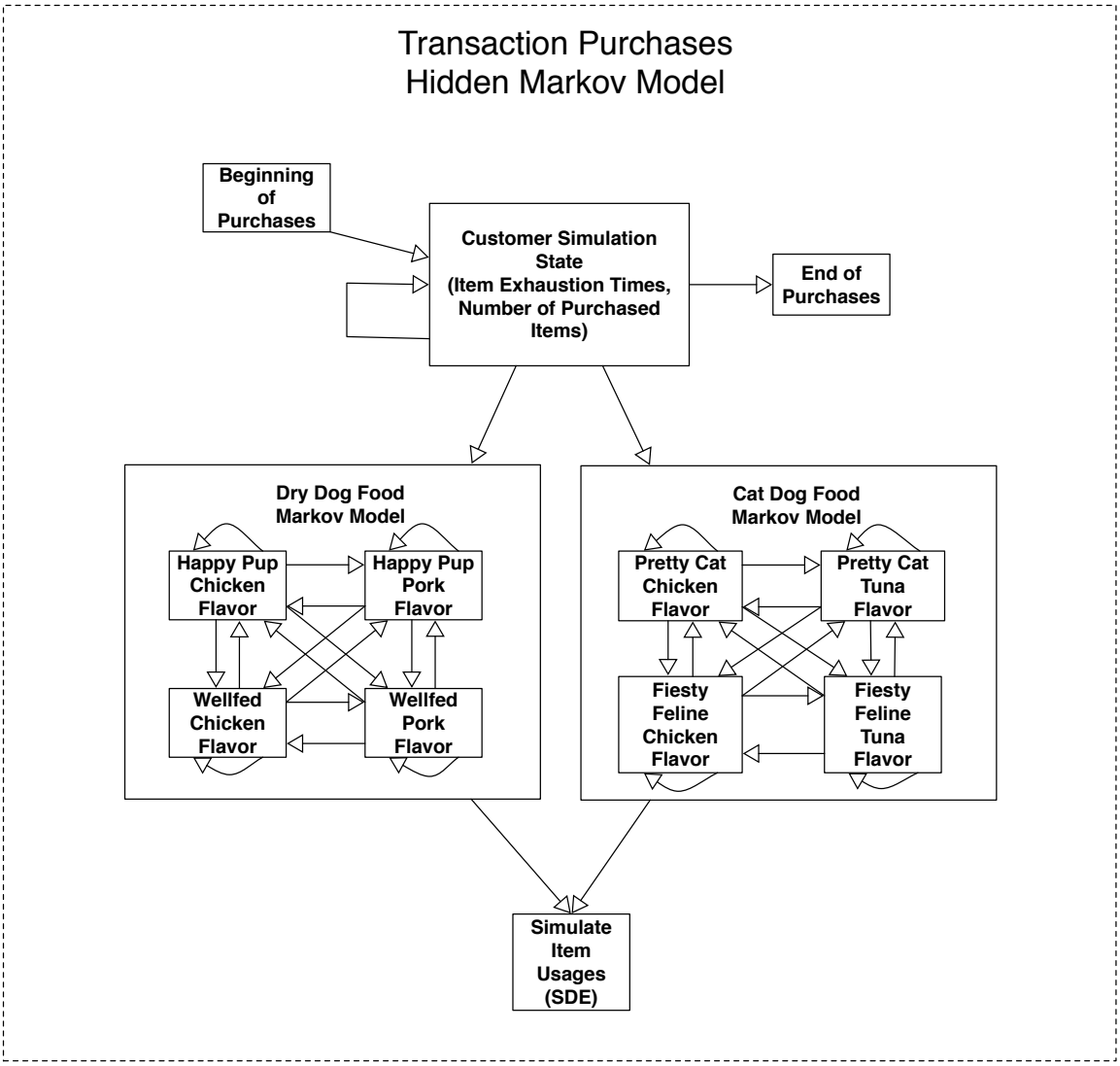

Fig. 3. Example Shopping Cart Hidden Markov Model

III. Implementation

The models and simulations were implemented using Python. The source code is available on GitHub [25] 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 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

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

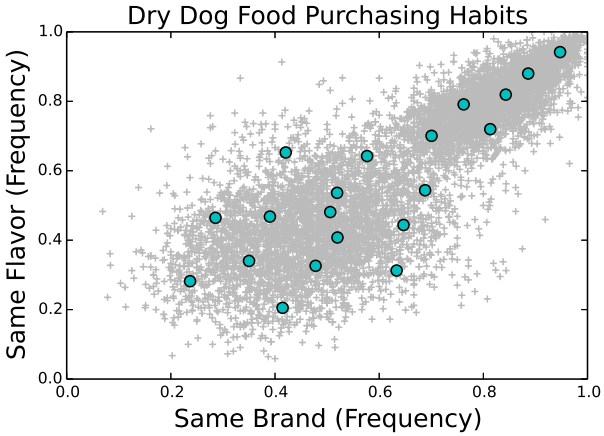

Fig. 5. Clustering of Brand and Flavor Purchasing Preferences. Grey plus signs (+) represent customer data points, and cyan circles represent cluster centers.

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, 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.

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

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

[2] 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.

[3] 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.

[4] 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.

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

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

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

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

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

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

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