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*—The abstract goes here.**

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, Pig and Crunch reference based around big data frameworks such as Hadoop, Storm, and Spark reference 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 rigorour 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 reference) offer access to their data but the costs and usage restrictions can be undesirable.

Ab initio mathematical models for generating synthetic data overcomes several of these obstacles. The code for generating the data is often small, enabling free hosting, licensing restrictions are often avoided, and the data size can be scaled, enabling seamless usage of the same data set across a range of scenarios. Common synthetic data set generators such as TeraGen and the Intel Hadoop Benchmark reference can generate data sets quickly and at any scale, but they are over-specialized for benchmarking, resulting in semantically-void data that lacks usefulness for analytics examples. The lack of frameworks for generating semantically-rich data at arbitrary scale denies developers access to consistent data sets that can be used across a range of environments Huppler, Pavlo.

Attempts have been made to find a middle-ground. Myriad and PDGF provide primitives for specifying generative data models reference. Alexandrov, et al. proposed Oligos, an approach for generating synthetic data from models trained on reference databases with real data reference. Oligos appears promising but will need to overcome several difficult challenges in automatically characterizing complex reference data to determine relevant constraints and learn accurate models. Until such methods have reached maturity, development of a customized domain-driven model is best suited for generating for satisfying the needs of big data software and algorithms development.

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 from across the Hadoop and Spark ecosystems reference. BigPetStore's usefulness has been demonstrated by its incorporation into the Apache BigTop distribution for testing and as a reference application reference.

In this work, we present a new domain-drive model and simulation for generating arbitrarily large, complex data for testing and benchmarking of software tools 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) reference and MovieLens (semantically rich) reference 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 apply it as a drop in replacement for the current version of BigPetStore which has already been utilized to test several Hadoop deployments at terabyte scale, and is actively being adopted by a variety of companies in the software industry, and confirm that it generates data with desirable properties (global patterns and region specific transactions).

II. Overview of Model & Simulation Procedures

need an intro paragrpah

A. Stores

Store records contain a unique identifier and a location in the form of a 5-digit zip code. The locations of the stores are modeled by a PMF that gives the probability that a store's location is the given zip code. We've written the PMF to give to zip codes in high-population, high-income areas the highest probabilities. The PMF is composed of two individual PMFs. One PMF determines the probability of each zip code as the population of that zip code over the total population:

$$p(\text{location} = z|\text{population}(z)) = \frac{\text{population}(z)}{\sum_i \text{population}(i)} \quad (1)$$

The second PMF 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:

$$p(\text{location} = z|\text{income}(z)) = s^{\left(\frac{\text{income}-\min_i \text{income}(i)}{\max_i \text{income}(i)-\min_i \text{income}(i)}-1\right)} \quad (2)$$

The combined PMF is given as:

$$\begin{aligned} p(\text{location} = z) =& Z^-1 p(\text{location} = z|\text{population}(z)) \\ & p(\text{location} = z|\text{income}(z)) \end{aligned} \quad (3)$$

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 PMFs are parameterized using external data sets. The list of zip codes came from a database of zip code latitude and longitudes. The population and household income data [reference] for the zip codes were taken from from the U.S. Census.

The stores' locations are generated by sampling zip codes from the PMF. A variety of methods can be used such as Roulette Wheel sampling and Monte Carlo methods.

B. Customers

Customer records consist of a unique identifier, a name, a location given as a 5-digit zip code, and the number and species of their pets.

The customers locations are modeled so that customers are placed within The customer location is modeled as an exponential distribution over the distance to the nearest store.

[discuss parameters used in later sections, methods for setting number of customers per store]

C. Transactions

A transaction model is created for each customer and integrated over the simulation's timeframe to generate transactions. Transaction records consist of a customer, a store, a date and time, and a list of items purchased and their prices. The store is set to the store located closest to the customer. The date and time is given as a floating-point number with units of days since the start of the simulation. (Timezones are not taken into account.) Transaction times are generated from a Monte Carlo process based on the customer will use up items such as dog food (Section II-C1). 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-C2).

1) Transaction Times: Transaction times are simulated using a Monte Carlo method (Figure 1). The usage over time of each category is simulated, from which transaction times are found for each item category. The earliest transaction time is taken as the 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 simple stochastic differential equation (SDE):

$$\frac{da_i}{dt} = -\mu_i + \sigma_i \, dW_i(t) \quad (4)$$

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 reference with timestep Δt_n sampled from a Poisson process:

$$\begin{aligned} & a_{n+1,i} = a_{n,i} - \mu_{i,c} \, \Delta t_n + \sigma_{i,c} \sqrt{\Delta t_n} R_n \\ & \Delta t_n \sim \text{Exp}(\lambda_i) \\ & R_n \sim \text{N}(0, 1) \end{aligned} \quad (5)$$

where λ_i is the item category's average usage rate, μ_i is the item category's average amount used per use, and σ_i^2 is the variance of the amount of the item category used per use. The parameter λ_i is defined for each item cateogry. The parameters

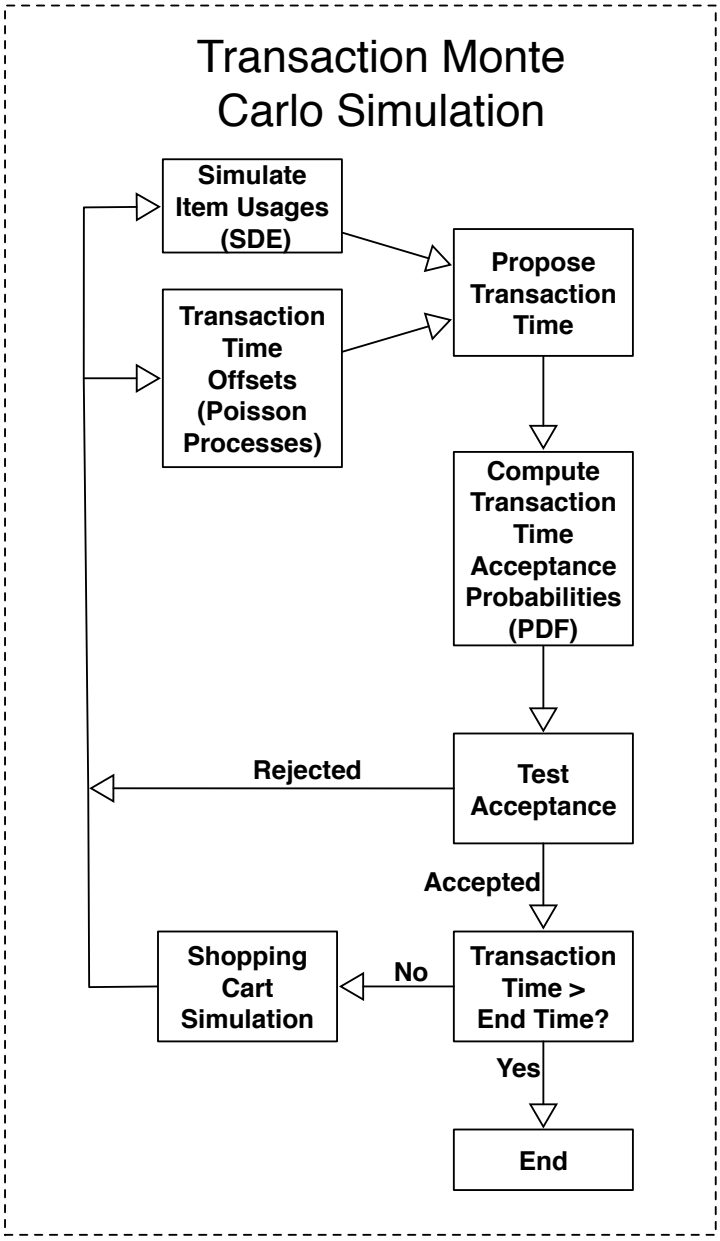

Fig. 1. Flowchart of Transaction Monte Carlo Simulation

$\mu_{i,c}$, and $\sigma^2_{i,c}$ 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. For each item cateogry, 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 earlist proposed transaction time is taken as the overall proposed transaction time T_T.

$$\beta_c \sim \mathrm{U}(a, b) \tag{6}$$
$$T_{O,i} \sim \mathrm{Exp}(\beta_c^{-1}) \tag{7}$$
$$T_{P,i} = T_{E,i} - T_{P,i} \tag{8}$$
$$T_T = \min_i \{T_{P,i}\} \tag{9}$$

The acceptance probability $p(t = T_T)$ of the proposed transaction time T_T is modeled using a PDF that incorporates factors such as store hours. The PDF could be extended to incorporate additional factors including time-dependent events such as sales, weather, and the amount of money the customer has available at the time.

$$p(t = T_T) = p(t = T_T | \text{store hours}) \tag{10}$$

The proposed transaction time is accepted if $p(t = T_T) < r$, 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 below. Transactions are generated until the accepted transaction time is later than the end time given as a simulation parameter.

2) Shopping Cart Items: A customers purchases in each transaction are modeled using a layered Hidden Markov Model (HMM) (Figure 2). The HMM has three 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 purchase states correspond to the item categories and are Markov models for choosing the item to be purchased from the category. The observables are assigned probabilities based on the amounts of time (based on the usage simulations) until the customer uses the remaining amount of each item category.

$$Pr(X_{n+1} = x | X_n) = \ldots \tag{11}$$

Markov Models are used to model the customer's purchasing behavior for each item category. Each state corresponds to an item in that category. The transition probabilities are assigned according to the customer's buying 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.

$$Pr(X_{n+1} = x | X_n) = \ldots \tag{12}$$

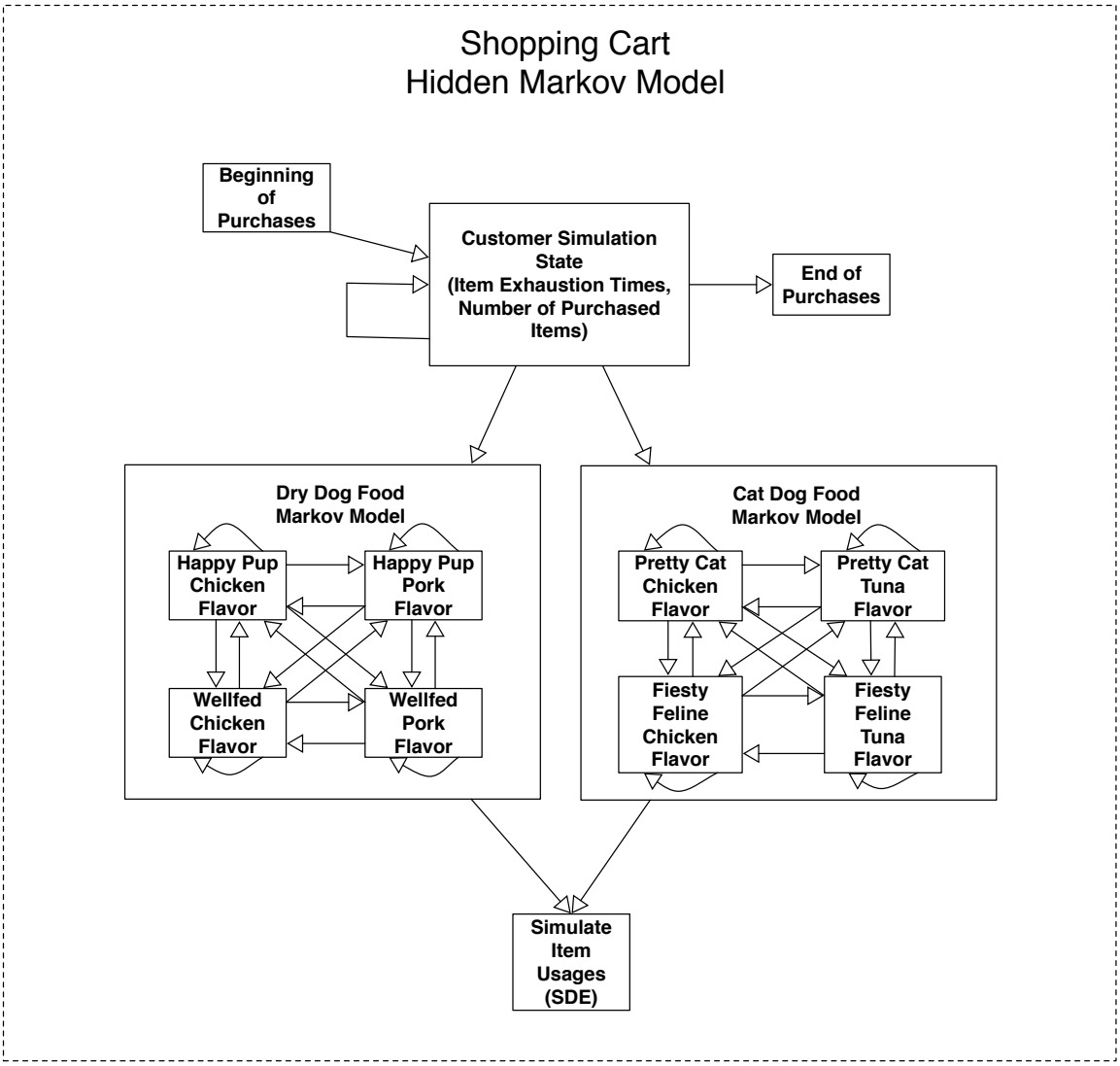

Fig. 2. Shopping Cart Hidden Markov Model

After an item is chosen, the item category's exhaustion time is updated by propogating the item category usage SDE described in Section II-C1. 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 reference at https://github.com/rnowling/bigpetstore-data-generator under the Apache 2 license.

IV. Example Analysis of Generated Data

Transaction data for 10 stores, 10,000 customers, and 4 years of simulated time totaling 100 MB was generated using the Python implementation.

A. Store Locations

[locations, population, median household income]

B. Customers

[distances to stores]

C. Transactions

* stores – distance, one or multiple stores? * brand loyalty

V. Scaling of Data Size and Run Time

BigPetStore aims to scale from a local desktop to a large cluster. As a result, we benchmarked the generation of data 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).

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

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

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.

VI. DISCUSSION AND CONCLUSION

implementation [want a modular framework with clearly-defined extension points, modularity so that users do not need to know about the machinations of the entire model, and so that modules are easy to develop and turn on/off dynamically]

[spark, hadoop ecosystem, parallelization of generating stores, customers, and transactions]

[model extensions]

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.