2016 International Conference on Internet of Things and Applications (IOTA)
Maharashtra Institute of Technology, Pune, India 22 Jan - 24 Jan, 2016

IoT and Distributed Machine Learning Powered Optimal State Recommender Solution

Mohit Sewak
Advanced Analytics Division
BNY Mellon Innovation Center
Pune, India - 411 028
Email: mohitsewak@gmail.com

Sachchidanand Singh
Business Analytics Division
IBM Software Lab
Pune, India - 411 057
Email: sach.success@gmail.com

Abstract— **Recommender systems add significant benefits to E-commerce in terms of sale conversion, revenues, customer experience, loyalty and lifetime value. But the recommendations from these systems do not change on inputs beyond user and item profile and transaction data.**

There have been some attempts in the past to optimize on more varied data in recommenders, example of which is the location based recommenders. But location is just one dimension of the state that a user could have shared with GPS/GLONASS/BaiDeu sensor available in most Smartphones. With an upcoming era of Smart-wears and pervasive IoTs, there are a lot many other dimensions of a user state which can be utilized to optimize upon the concept of Optimal State Recommender Solutions.

This paper suggests upgrading from conventional recommendations that are based on user/ item preferences alone with systems that provide the best recommendation at the most optimal state when the user is most receptive to accept the recommendation, the "optimal state recommendation solution" and proposes solutions and architectures to overcome the challenges of dealing with real time, distributed machine learning on IoT scale data in implementing this solution. The paper leverages some of the advance distributed machine learning algorithms like variants of Distributed Kalman Filters, Distributed Alternating Least Square Recommenders, Distributed Mini-Batch Stochastic Gradient Descent(SGD) based Classifiers, and highly scalable distributed computation and machine learning platforms like Apache Spark, (Apache) Spark MLlib, Spark Streaming, Python/PySpark, R/SparkR, Apache Kafka in an high performance, distributed, fault tolerant architecture. The solution also aspires to be compliant with upcoming IoT standards and architectures like IEEE P2413 to provide a standard solution for such problems beyond the current scope of this paper.

Keywords—IoT, Distributed Machine Learning, Apache Spark, MLlib, PySpark, SparkR, Apache Kafka, Recommender System, Distributed Kalman Filter, Distributed ALS Recommender, Mini Batch SGD, IEEE P2413, Smartwears, Sensors.

I. INTRODUCTION

The existing recommendation systems are based on user preference only [1], [2]. Though there are some location based [2] but there are no (multi-dimensional) state based recommendation system available in public literature.

The effectiveness of recommendations can be improved by using state, time context & group preferences. The same recommendation at an unsuitable state or time may not be effectively converted into actual purchase. Moreover, such recommendations when repeated, pose a greater risk of turning the user off from such push notifications and losing his share of attention even on a genuine recommendation, which otherwise could have resulted in a purchase. Moreover, since most recommendation system takes quite a while to settle down for any new user, such ineffectiveness of the mode of recommendations, combined with the inefficiencies due to lack of data for the new user pose a great risk of losing a customer permanently instead of increasing his share of attention and wallet.

The customer's propensity to buy a particular recommended item does not always remain the same [3], [4] across all states (a multi-dimensional concept of user state, also including time as one of the dimension). We can increase the probability of acceptance of targeted items offered to customer by sending a recommendations to the targeted customers when they are most likely to give positive response to these push recommendations through Smartphones or any push enabled recommendation channel (inducing state based impulse for the product-offer combination).

For example an E-commerce grocery player 'A', wants to increase his market share, and has been sending e-mail/ SMS/ app-based notifications of discounts on targeted products to the customers. But such recommendations evades customer attention and after some time customer stops bothering about them. Whereas another E-commerce grocery player 'B' is using Recommender solutions proposed in this paper. Player 'B' can now send a discount offer on a health drinks just when the customer has finished an extraneous workout, and is able to communicate and associates the discount offered with the quantum of physical work out. Player 'B', in this case has an opportunity to change the habits of customers in a positive manner, and able to get positive attention of the customer since his notifications are directly tied with the user's behavior. Similar concepts can be applied to insurance or any other industry as well.

978-1-5090-0044-9/16/$31.00 ©2016 IEEE

II. Brief Introduction of Algorithm and Technology Used

A. Algorithms

We would require implementing distributed variants from three different families of algorithms. The first one being algorithms for reducing noise from sensor data. Algorithms based on recursive bayes techniques like Kalman filters are very popular in this application. For our solution, owing to the scale and speed at which we are expecting the data, the linear and more popular version of Kalman Filter will pose a serious bottleneck. There are some implementations Distributed variants of Kalman Filters in literature [15],[43]. We will be using one of these for the purpose of our solution.

Once we bring the real time sensor data, treat it for noise, it is ready for feature generation and general manipulation. Most of this work can be done using spark's/ PySpark's API. We would require a real time distributed classification algorithm implementation next. Gradient descent is a popular optimization algorithm for classification problem. It has both the batch and the stochastic/ real time (SGD) versions. We would be using the distributed versions of stochastic gradient descent, namely the Distributed Stochastic Gradient Descent (DSGD) [21] for our solution. A lot of different variants of this, including the mini-batch stochastic gradient descent have also been implemented on the Spark platform.

Next we would require a distributed Recommender algorithm. Recommendation generation is mostly an offline task, which could be done (and periodically refreshed) in the background. But the computational complexity of recommendation algorithm, coupled with IoT scale data, would render most of the existing (user/item) non-distributed algorithms recommender system algorithms bottleneck for the solution. Therefore we would use a very scalable and distributable recommendation algorithm based on Alternating Least Square (ALS) method[5]. ALS based recommenders are also available out of box in Spark's MLlib, and has API in python as well.

B. Technology

The proposed solution starts with mobile/smart-wear devices sending data to cloud based Message Queuing Telemetry Transport (MQTT) servers. MQTT is a machine-to-machine (M2M) connectivity protocol used in Internet of Things (IoT) [8], [27], [28]. With the mobile devices uploading to these cloud servers, the extremely light weight publish & subscribe architecture of MQTT would be useful as it can work well with minimal code footprint and network bandwidth requirements, with devices connected remotely [8].

From (cloud based) MQTT onwards things need to scale up to distributed, real time environments Apache Kafka is a fast, scalable, fault-tolerant, cloud compatible and durable queuing and messaging environments to feed other high throughput computational and machine learning platforms. Apache Kafka is a very capable solution offering these capabilities [6]. Besides providing good message broking capabilities, Apache Kafka also has good Hadoop connectivity. Apache Kafka, developed by LinkedIn, is a high throughput, distributed messaging platform, also providing persistence of messages, Kafka's consumer can even do an off-sync analysis by rewinding a stream of events and also allows efficient bulk dump of data into HDFS for offline analysis [7].

Apache Storm and Apache Spark (Streaming) are two very potent distributed CEP (continuous event processing) engines, for connecting to Apache Kafka. Owing to the eco system that Spark brings with itself (besides the Spark Streaming), a lot of modern architectures and even bigger players (like twitter) seems to be favoring Spark based solutions. In our case, since we need to leverage machine learning, and distributed computation (beyond streaming) as well, we would go with Apache Spark in our solution. Also the rich Python and R based API's of Apache Spark, helps bridge the data scientist and solution engineer divide.

From the Apache Spark eco system, besides the Spark Streaming, we would also utilize the core spark context (sc) for general computation and feature creation, and MLlib (Machine Learning Library). MLlib consists of common distributed machine learning algorithms and other utilities [9]. PySpark is the Python API for Spark, and is available since Spark release 0.7.0 [10]. It is built on top of Spark's Java API [13], and the data is processed in Python and cached in JVM. Spark Context uses Py4J library to launch a JVM and create a Java SparkContext in Python driver program. PySpark supports Spark features like RDD (resilient distributed dataset) accumulators, broadcast variables and HDFS integration.

III. Distributed Machine Learning And Recommender System Algorithms

We will use distributed Kalman filters, distributed ALS (Alternating Least Square) recommender and distributed mini batch SGD (Stochastic Gradient Descent) for our solution.

A. Distributed Kalman Filter

The Distributed Kalman filters (DKF) addresses the noise reduction problem of sensor data in real time. This algorithm breaks it into two parts in terms of weighted measurements & inverse-covariance matrices and solved using low-pass & band-pass consensus filters [16].

1) *Low Pass Consensus Filter(CF$_{lp}$):* If qi denote m-dimensional state of node i and u_i denote the m-dimensional input of node i. Then dynamic consensus algorithm [16]-

$$\dot{q}_i = \sum_{j \in N_i} (q_j - q_i) + \sum_{j \in N_i \cup \{i\}} (u_j - q_i)$$

This can be written in mathematical expression as:

$$\dot{q} = -\hat{L}q - \hat{L}u + (I_n + \hat{A})(u - x)$$

With $q = \mathrm{col}(q_1, \ldots, q_n)$, $\hat{A} = A \otimes I_m$ and $\hat{L} = L \otimes I_m$ gives a low-pass consensus filter with a multi-input/multi-output (MIMO) transfer function-

$$H_{lp}(s) = [(s+1)I_n + \hat{A} + \hat{L}]^{-1}(I_n + \hat{A})$$

from input u to output x.

It is used for fusion of measurements which calculates $\hat{y}_i$ by applying the algorithm to $HiR{-}i\,zi$ the input of node i [16].

2) Band Pass Consensus Filter(CF$_{bp}$): This filter can be shown as below mathematical expression [16],[17]:

$$H_{bp}(s) = H_{lp}(s)\, H_{hp}(s)$$

This can be equivalently stated in the form of a dynamic consensus algorithm

$$\dot{e}_i = -L\hat{}\, e_i - \hat{}\, Lu_i,$$
$$p_i = e_i + u_i,$$
$$\dot{q}_i = \sum_{j \in N_i} (q_j - q_i) + \sum_{j \in N_i \cup \{i\}} (p_j - q_i)$$

With a state $(e_i, q_i) \in R^{2m}$, input u_i, and output q_i.

It calculates $\hat{}\, Si$ column-wise for node i by using filter on columns of $H_{-i} R_{-1i} Hi$ as the inputs of node i. [16], [17].

B. Distributed ALS Recommender

Most Recommender system algorithms based on user/item similarity suffers from two problems, the scarcity of the user - item rating matrix (since not all users have ratings for all items, and not all items have ratings given them, finding similar users or items is difficult and memory-intensive), and the cold-start problem, which refers to the initial lack of information from which reliable recommendations could be made. To mitigate these problems, hybrid approaches are often used [18]. A common problem faced by internet companies like Netflix or Amazon is to recommend new products to users in personalized settings. It can be formulated as a learning problem in which we have ratings given by users for certain items and we are predicting user's rating for the remaining items.

If there are n users and m items then we have n x m matrix R in which the (u, i)th entry is r_{ui} rating for item i by user u. The missing entries in matrix R indicates unobserved ratings and we have to estimate these unobserved ratings [19]. We can distribute the computation of the ALS algorithm in several ways like join or broadcast method [20].

1) Join Method[19]: We consider a fully distributed version where all the data (ratings) and parameters (X and Y) are distributed and stored in Resilient Distributed Datasets (RDD).

$$\text{Ratings} : \text{RDD}((u; i; r_{ui}); : : :)$$

The ratings are always sparse and we use dense representation for factor matrices X and Y which are stored as Resilient Distributed Datasets (RDD) of vectors:

$$X : \text{RDD}(x_1; : : : ; x_n)$$
$$Y : \text{RDD}(y_1; : : : ; y_m)$$

We know ALS algorithm for Matrix Completion-

$$x_u = (\sum_{r_{ui} \in r_{u*}} y_i \, y_i^T + \lambda I_k)^{-1} \sum_{r_{ui} \in r_{u*}} r_{ui} \, y_i$$

To compute part A $\{ (\sum_{r_{ui} \in r_{u*}} y_i \, y_i^T + \lambda I_k)^{-1} \}$ and part B $\{ \sum_{r_{ui} \in r_{u*}} r_{ui} \, y_i \}$, we can follow the below steps:

a) Join Ratings with Y factors using key i (items)

b) Map to compute $y_i y_i^T$ and change key to u (user)

c) ReduceByKey u (user) to compute $\sum_i y_i y_i^T$

d) Invert

e) Another ReduceByKey u(user) to compute $\sum r_{ui} y_i$

Similarly we can compute below mathematical equation-

$$y_i = (\sum_{r_{ui} \in r_{*i}} x_u x_u^T + \lambda I_k)^{-1} \sum_{r_{ui} \in r_{*i}} r_{ui} \, x_i$$

2) Broadcast Method [19]:

a) Create two copies of ratings with different partionings- the ratings by user are partitioned to create R1 and ratings by item to create R2. Also in R1 all ratings by the same user are on the same machine and in R2 all ratings for same item are on the same machine.

b) Broadcast X; Y

c) We can compute the update of x_u locally on each machine using R1 and Y -

$$x_u = (\sum_{r_{ui} \in r_{u*}} y_i \, y_i^T + \lambda I_k)^{-1} \sum_{r_{ui} \in r_{u*}} r_{ui} \, y_i$$

d) We can compute the update of y_i locally on each machine using R2 and X-

$$y_i = (\sum_{r_{ui} \in r_{*i}} x_u x_u^T + \lambda I_k)^{-1} \sum_{r_{ui} \in r_{*i}} r_{ui} \, x_u$$

It can be further optimized by making group of X and Y factors into blocks i.e. user blocks and item blocks and reduce the communication by only sending the block of users(or items) to each machine that are needed to compute the updates at that machine. It is known as Block ALS.

C. Distributed Stochastic Gradient Descent (DSGD) algorithm

The exact factorization is practically not possible and all matrix factorization algorithms attempts to minimize loss function but they give low rank approximations [21]. The recent development of parallel processing frameworks like MapReduce has made web-scale matrix factorizations quite feasible for enterprise dealing with large amount of data. The Distributed Stochastic Gradient Descent (DSGD) algorithm have good performance in non-parallel environments [22].

We will apply Stochastic Gradient Descent(SGD) to matrix factorization. If $\theta = (W, H)$ and decompose the loss L as shown in below mathematical equation

$$L = \sum_{(i,j) \in z} l(V_{ij}, W_{i*}, H_{*j})$$

for training set Z and local loss function l, we can show local loss at position $z = (i,j)$ by $L_z(\theta) = L_{ij}(\theta) = l(V_{ij}, W_{i*}, H_{*j})$. Therefore by sum rule for differentiation $L'(\theta) = \sum_z L'_z(\theta)$The algorithm given below is using Stochastic Gradient Descent(SGD) to perform matrix factorization-

Algorithm: Stochastic Gradient Descent(SGD) for Matrix Factorization[21]

Require: A training set Z, initial values W_0 and H_0

while not converged **do** /* step */

Select a training point $(i, j) \in Z$ uniformly at random.

$W'_{i*} \leftarrow W_{i*} - \varepsilon_n N\{\partial/(\partial W_{i*}\} \, l(V_{ij}, W_{i*}, H_{*j})$

$H_{*j} \leftarrow H_{*j} - \varepsilon_n N\{\partial/(\partial H_{*j}\} \, l(V_{ij}, W_{i*}, H_{*j})$

$W_{j*} \leftarrow W'_{j*}$

end while

We need to update W_{i*} and H_{*j} after selecting a random training point (i,j) ∈Z-

$$\{\partial/\partial W_{i'k}\}\ L_{ij}(W,H) = \begin{cases} 0 & \text{if } i \neq i' \\ (\partial/\partial W_{ik})\ l(V_{ij}, W_{i*}, H_{*j}) & \text{otherwise} \end{cases}$$

and

$$\{\partial/\partial H_{kj'}\}\ L_{ij}(W,H) = \begin{cases} 0 & \text{if } j \neq j' \\ (\partial/\partial H_{kj})\ l(V_{ij}, W_{i*}, H_{*j}) & \text{otherwise} \end{cases}$$

Its beneficial to replace exact gradients (GD) by noisy estimates (SGD) since calculation of exact gradient is costly while noisy estimates are quick and easy to obtain. It has been observed in case of large-scale matrix factorization that increased number of steps leads to faster convergence [22],[24].

IV. Proposed Solution And Architecture

This paper suggests an architecture to build a recommender system which provides personalized recommendations for products and services to consumers based on their most optimal state to respond to the recommendations. The architecture for the proposed optimal state based recommender solution is shown in Fig. 1.

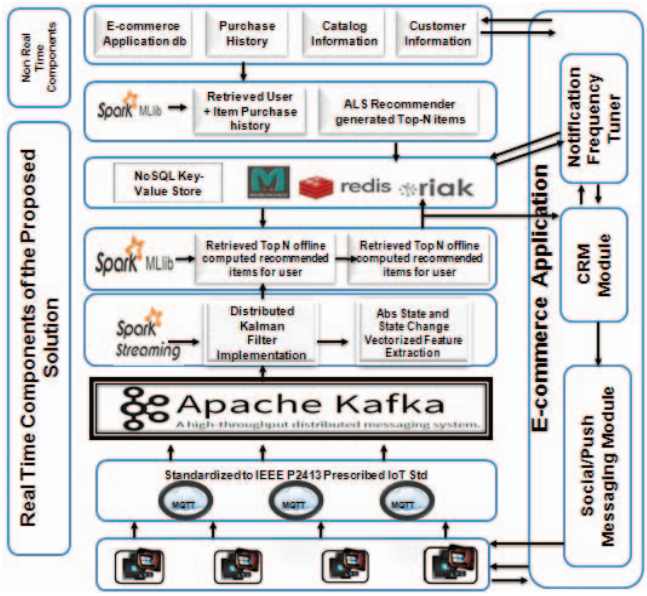

Fig. 1. Reference Architecture for Optimal State Based Recommender Solution

User's absolute state comprising of the measures of all the dimensions capture-able using mobile/ smart-wear sensors (location, acceleration, heart beat etc.), events (device active, sleep etc.) and global data (time etc.) can be captured in real time and sent to location specific cloud based MQTT servers. Such location specific cloud based MQTT servers, besides enhancing performance, can also deal with localization specific nuances in data or device types.

The output from these cloud based MQTT servers is standardized for upstream consumptions, in order to isolate the data and device specific difference and enable generalization of the solution across multiple locations, devices and business/ industry types. Currently, most of the IoT solutions are impacted because of these issues leading to a lot of redundant work and sub-optimal solutions. As of the writing of this paper, IEEE has a working group (P2413), which is working on the Standard for an Architectural Framework for the Internet of Things [25],[31], which also aims to provide a reference blueprint for sensor data abstraction and quality [26],[32]. The output from the MQTT can be standardized to this blueprint standard.

So far the data has been distributed across multiple local MQTT instances, but for a powerful learning solution, we need to integrate this data. To achieve this, all the individual MQTT instances may be made to act as Apache Kafka's publishers, publishing to a standard or customized Kafka topics. The topics can be differentiated by user type, location type, usage type, device type, data dimension type etc. for custom upstream solution and data models. Kafka works well with distributed stream processing consumers like Apache Storm and Apache Spark[33],[34], and has multiple proven high performance solutions based on this coupling [29],[12],[14], [30],[35],[36],[37],[38].

For our solution we would use Apache Spark Streaming as a consumer to Kafka because of the synergies of computational and machine learning ecosystem that comes with Spark [16], [39], and its rich Python [18],[40] and R [21],[41] based APIs.

With the data in motion, in Apache Spark streaming, real time, distributed Kalman Filters, which is implementable in Spark/PySpark [16],[19],[22],[23],[24],[42],[44] can be applied. Most sensor data in raw state is inaccurate and not conducive for any analytical or machine learning application. Kalman Filters help optimize this data for upstream analytical processing.

Besides Kalman Filters, Spark streaming can also be used to vectorized the user state and to extract important features on which good predictive algorithms can be based. The filtered user state data in absolute state might not be informative enough, and useful transformations like extracting direction (source-destination) of approach, distance to key destinations, speed, heart beat as percentage of peak heart beat rate, rate of change of heart beat count etc might be more insightful. This transformation can again be implemented in Spark Streaming's Scala (or Java) or Python APIs [20],[45].

One of the key challenge of this solution is to predict in real time when (from a multi-dimensional state perspective) a particular recommendations need to be delivered to a specific user. This part of the problem is broken down into three parts. The first part constitutes generating the top N recommendations for a specific user. This is can be done offline using distributed recommendation algorithms like regularized ALS Recommenders, and are already available in Apache Spark MLlib[1],[46]. Other Collaborative filtering algorithms can also be used, but linear algorithms may not be scalable under such conditions and some framework optimized distributed recommender algorithms would be more optimal. The user, customer, and order data for this algorithm can be sourced from the respective e-Commerce application databases, and the user-recommendation (key-value pairs) obtained from this step can be stored in a high performance /distributed (key value) store readily accessible in real time for the algorithms powering the second (real time) part of the problem.

The second part of the problem involves determining the state specific recommendation score of some of the most likely available recommendations (as computed in earlier step)

for the specific user state in real time as and when user state data (and computed features) are obtained from streams. For the purpose of this part of the solution we would require a distributed, real time classification algorithm which could classify the user state into purchase propensity for the available item choices. For this purpose also we would use a framework optimized distributed classification algorithm. The distributed stochastic (mini-batch) gradient descent (DSGD) based (SVM [2],[47] or logistic [3],[48]) classifier available in Apache Spark MLlib would be good for the purpose. There are other custom third party implementations like budgeted mini-batch gradient descent also available for Spark [4],[49]. The state based item purchase probability from the classification algorithm can be combined with the standardized item recommendation score for the specific user to arrive at a standardized optimal state item recommendation score for the user at that particular state as:

E(Item | F(User, V(State))) ~ F(E(Item | R(User)) , E(Item | V(State)))

Where E(Item | F(User, V(State))) is the expectancy of an Item's preference by a specific User in a given State as represented by the state feature vector V(State), E(Item | R(User)) is the recommendation score for the specific item for the given user, E(Item | V(State)) is the purchase propensity a particular item the specific user state as represented by the state feature vector V(State). Though many relevant versions of this function can be scientifically formulated, for the purpose of the paper let's assume a simplistic scaled/ standardized version as below.

E(Item | F(User, V(State)))scaled = S_f * F(E(Item | R(User)) * E(Item | V(State)))

Where S_f is the scaling factor for the user item combination as obtained from the E(Item | F(User, V(State)))raw from the current snapshot of the optimal state recommendation database as below:

E(Item | F(User, V(State)))scaled(User=x, Item=i) = (E(Item | F(User, V(State)))raw(u, i) – Min(E(Item | F(User, V(State)))raw))/ (Max(E(Item | F(User, V(State)))raw))- Min(E(Item | F(User, V(State)))raw)))

Where E(Item | F(User, V(State)))scaled(User=x, Item=i) would be referred to as optimal state recommendation score for recommending a the specific item to a given user in a given state, or simply OSR score.

In the third part of the solution it is determined whether the given OSR score warrants an actual push notification of a recommendation been sent to the user on his mobile device or through other push channels where the user can receive the notification in a state, while still the user state is not very different from the state for which the OSR score was computed. Remember that we are getting tons of data, and almost at every instance the user device is sending the data. So if we do not have such mechanism to send the recommendations only for very specific moments/ states, we will end up sending too many recommendations, the result of which could be as counterproductive as highlighted in the Introduction part of the paper. Whether a recommendation with a given OSR score is to be sent is based on multiple factors like optimal custom OSR score threshold (for the user, item or user-item combination), previous recommendation time gap for a similar product sent to the user, and custom user preferences for notifications. All this data is stored in the data store, and rules using this data can be configured for different product/ user classes in the "Notification Frequency Tuner" module of the solution, which is coupled to the CRM module on the application. The CRM module receives the OSR score intimation for each user state (which can be further used for driving other intelligent solutions), but the push notification for the specific notation is sent only when the "Notification Frequency Tuner" approves a notification.

V. Conclusion And Scope For Future Work

In this paper we presented architecture, solution, and algorithms to enable the concept of Optimal State based Recommender (OSR) System powered by IoT, personalized smart wear, on scalable, fast, high performance, distributed messaging, computing, machine learning framework and eco-system. The OSR concept opens new venue for customized marketing for modern e-Commerce companies in an era where most of the existing multi-channel marketing and recommendation push methods are no more the differentiating factor for modern technology start-ups and established players. Besides higher sales conversion, it may also help drive higher customer satisfaction & loyalty by an essential influencer of customer habits and lifestyle.

The proposed solution, architecture, and algorithms could be applied to multiple industries trying to explore IoT and distributed machine learning and big data based sales and marketing solutions. All the components in the architecture are open source, and can run on commodity hardware on premise or in cloud, leading to a highly flexible, scalable and low cost solution, that could be elastically adapted to the company's current scale dynamically. The algorithms come in different flavors and could also be further customized depending upon the variations in the desired outcomes and the data.

As a scope for further work, the next logical step would be to quantifying and comparing the effectiveness of the OSR system with an incumbent recommender solution and coming up with factors influencing the effectiveness in either direction. Another area of value addition would be to come up with more optimal variants of the algorithms used to empower the concept, and implement them on the chosen distributed computing/machine learning frameworks and integrate it with the other parts of the solution.

References

[1] Jiliang Tang; Xia Hu; Huan Liu, "Social Recommendation: A Review", http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.431.6335&rep=rep1&type=pdf

[2] Jie Bao; Yu Zheng; Mohamed F. Mokbel, "Location-based and Preference- Aware Recommendation Using Sparse Geo-Social Networking Data", http://research.microsoft.com/pubs/172445/LocationRecommendation.pdf

[3] Ofer Mintz; Imran S. Currim; Ivan Jeliazkov, "Information Processing Pattern and Propensity to Buy: An Investigation of Online Point-of-Purchase Behavior", http://www.economics.uci.edu/~ivan/MCJforthcoming.pdf

[4] Firebrick consulting , "Propensity-to-Buy Segmentation", http://www.firebrickconsulting.com/StrategiestoRevenue/category/propensity-to-buy-segmentation/

[5] Mahout , "Introduction to ALS Recommendations with Hadoop", https://mahout.apache.org/users/recommender/intro-als-hadoop.html

[6] Hortonworks, "Apache Kafka", http://hortonworks.com/hadoop/kafka/

[7] Data Ingestion, "The Hadoop Ecosystem Table", https://hadoopecosystemtable.github.io/

[8] Dave Locke , "MQTT: Enabling the Internet of Things", https://www.ibm.com/developerworks/community/blogs/c565c720-fe84-4f63-873f-607d87787327/entry/tc_overview?lang=en

[9] Srini Penchikala , "Big Data Processing with Apache Spark – Part 1: Introduction", http://www.infoq.com/articles/apache-spark-introduction

[10] Spark, "Spark Release 0.7.0", http://spark.apache.org/releases/spark-release-0-7-0.html

[11] Spark 0.9.0, "Python Programming Guide", https://spark.apache.org/docs/0.9.0/python-programming-guide.html

[12] Avkash Chauhan, "Machine Learning Libraries in Python", https://cloudcelebrity.wordpress.com/2012/04/25/machine-learning-libraries-in-python/

[13] Joshua Rosen, "PySpark Internals", https://cwiki.apache.org/confluence/display/SPARK/PySpark+Internals

[14] Spark 1.1.0, "Machine Learning Library (MLlib)", https://spark.apache.org/docs/1.1.0/mllib-guide.html

[15] A. Abdelgawad, M. Bayoumi , "Low-Power Distributed Kalman Filter for Wireless Sensor Networks", http://www.jes.eurasipjournals.com/content/pdf/1687-3963-2011-693150.pdf

[16] Olfati-Saber, R., "Distributed Kalman Filter with Embedded Consensus Filters," in Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on , vol., no., pp.8179-8184, 12-15 Dec. 2005

[17] Reza Olfati-Saber, "Distributed Kalman Filtering and Sensor Fusion in Sensor Networks", http://link.springer.com/chapter/10.1007/11533382_10

[18] Rubing Duan; Goh, R.S.M.; Feng Yang; Yong Kiam Tan; Valenzuela, J.F.B., "Towards building and evaluating a personalized location-based recommender system," in Big Data (Big Data), 2014 IEEE International Conference on ,vol., no., pp.43-48, 27-30 Oct. 2014

[19] Reza Zadeh; Databricks and Stanford , "Matrix Completion via Alternating Least Square(ALS)", http://stanford.edu/~rezab/dao/notes/lec14.pdf

[20] Kenneth L. Clarkson and David P. Woodruff. 2013. Low rank approximation and regression in input sparsity time. In Proceedings of the forty-fifth annual ACM symposium on Theory of computing (STOC '13). ACM, New York, NY, USA, 81-90.

[21] Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis. 2011. Large-scale matrix factorization with distributed stochastic gradient descent. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '11). ACM, New York, NY, USA, 69-77.

[22] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. IEEE Computer, 42(8):30–37, 2009.

[23] Christopher Bishop, Pattern Recognition and Machine Learning, Information Science and Statistics, Springer-Verlag New York, 2006

[24] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In NIPS, volume 20, pages 161–168. 2008.

[25] Toby Jaffey , MQTT and CoAP, IoT Protocols, http://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php

[26] HiveMQ Enterprise MQTT Broker, MQTT 101 – How to Get Started with the lightweight IoT Protocol, http://www.hivemq.com/how-to-get-started-with-mqtt/

[27] InfoWorld , How to use MQTT for IoT messaging, http://www.infoworld.com/article/2957366/internet-of-things/how-to-use-mqtt-for-iot-messaging.html

[28] IBM developerWorks®, 5 Things to Know About MQTT – The Protocol for Internet of Things, https://www.ibm.com/developerworks/community/blogs/5things/entry/5_things_to_know_about_mqtt_the_protocol_for_internet_of_things?lang=en

[29] Spark 1.5.1 , SparkR (R on Spark), https://spark.apache.org/docs/latest/sparkr.html

[30] Apache Spark™ , Spark MLlib, http://spark.apache.org/mllib/

[31] IEEE Standards Association, P2413-Standard for an Architectural Framework for the Internet of Things(IoT), https://standards.ieee.org/develop/project/2413.html

[32] IEEE Standards Association , IoT Architecture-Internet of Things (IoT) Architecture, https://standards.ieee.org/develop/wg/IoT_Architecture.html

[33] Spark 1.5.1 , Spark Streaming + Kafka Integration Guide, http://spark.apache.org/docs/latest/streaming-kafka-integration.html

[34] Hortonworks®, Storm and Kafka Together: A Real-time Data Refinery, http://hortonworks.com/blog/storm-kafka-together-real-time-data-refinery/

[35] zData Inc, Real Time Streaming with Apache Storm and Apache Kafka, http://zdatainc.com/2014/07/real-time-streaming-apache-storm-apache-kafka/

[36] Michael G. Noll, Integrating Kafka and Spark Streaming: Code Examples and State of the Game,http://www.michael-noll.com/blog/2014/10/01/kafka-spark-streaming-integration-example-tutorial/

[37] zData Inc, Real Time Streaming with Apache Spark, http://zdatainc.com/2014/08/real-time-streaming-apache-spark-streaming/

[38] GitHub, Example integration of Kafka, Avro & Spark-Streaming on live Twitter feed, https://github.com/mkrcah/scala-kafka-twitter

[39] Mohit Sewak; Sachchidanand Singh, "A Reference Architecture and Road map for Enabling E-commerce on Apache Spark", Communications on Applied Electronics 2(1):37-42, June 2015. Published by Foundation of Computer Science, New York, USA.

[40] Spark, Welcome to Spark Python API Docs! http://spark.apache.org/docs/latest/api/python/

[41] R frontend for Spark, Documentation for package 'SparkR' version 1.5.1, https://spark.apache.org/docs/latest/api/R/

[42] Alexander Aprelkin, Master's thesis: Short Term Household Electricity Load Forecasting Using a Distributed In-Memory Event Stream Processing System, http://www.academia.edu/7817556/Masters_thesis_Short_Term_Household_Electricity_Load_Forecasting_Using_a_Distributed_In-Memory_Event_Stream_Processing_System

[43] The Kalman filter - Scala for Machine Learning, https://www.packtpub.com/packtlib/book/Big-Data-and-Business-Intelligence/9781783558742/3/ch03lvl1sec24/The%20Kalman%20filter

[44] PyData: Presentation Abstracts, http://pydata.org/berlin2014/abstracts/

[45] Spark 1.5.1, Spark Streaming Programming Guide, http://spark.apache.org/docs/latest/streaming-programming-guide.html

[46] Spark 1.5.1, MLlib - Collaborative Filtering, http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html

[47] SVMWithSGD,http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.classification.SVMWithSGD

[48] LogisticRegressionWithSGD , http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.classification.LogisticRegressionWithSGD

[49] Hang Tao; Bin Wu; Xiuqin Lin, "Budgeted mini-batch parallel gradient descent for support vector machines on Spark," in Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on , vol., no., pp.945-950, 16-19 Dec. 2014