Big Data Ingestion and Lifelong Learning Architecture

Gautam Pal
Department of Computer Science
University of Liverpool
Liverpool, UK
gautam.pal@liverpool.ac.uk

Gangmin Li
Department of Computer Science
Xi'an Jiaotong-Liverpool University
Suzhou, China
Gangmin.Li@xjtlu.edu.cn

Katie Atkinson
Department of Computer Science
University of Liverpool
Liverpool, UK
K.M.Atkinson@liverpool.ac.uk

***Abstract*—Lifelong Machine Learning (LML) mimics common human learning experiences. Humans undergo through long learning phase at start while studying followed by updating knowledge base incrementally from everyday instances. The objective is to retain past learnt knowledge and transfer learning to the next task iteratively. Training on the large data pool through a one-shot long running batch job limits the responsiveness and increases the infrastructure cost through large cluster requirements. The full dataset may not be available as well at the initiation of the training process. Through a review of previous work on lifelong machine leaning, we propose a Multi-agent Lambda Architecture (MALA) model to combine historical batch data with live streaming data to develop a lifelong learning system. MALA allows the streaming process to initialize itself with trained model from the batch data. Streaming process takes the batch data offset and incrementally updates the model iteratively with new waves of data. Reasons for our claim are presented through implementation of a recommender engine.**

***Index Terms*—Lifelong learning, Incremental learning, Multi-agent System, Recommender systems**

I. Introduction

Over the past two decades significant advancements has been made in the field of machine leaning frameworks and methods. However, not much research has been done on the framework that leverages these methods to retain and iteratively extend the past learnt knowledge over the period of time.

This work provides an approach towards developing an LML system [1] RN2 through big data hybrid data processing Lambda Architecture which consolidates the stream and batch views using big data tools. In our model batch and real-time components act as autonomous Multi-agent systems in collaboration. We propose a novel Multi-agent Lambda Architecture (MALA) model and prove the efficiency through implementation of a recommender service.

II. Proposed Architectures

A. Lambda Architecture

Lambda Architecture (LA) [1] is a standard framework for managing big data which enables mixing of real-time with batch data. The basic architecture of lambda offers three layers: speed layer for real-time data, batch layer for large volume of static historical data pool and serving layer that integrates real-time and batch views.

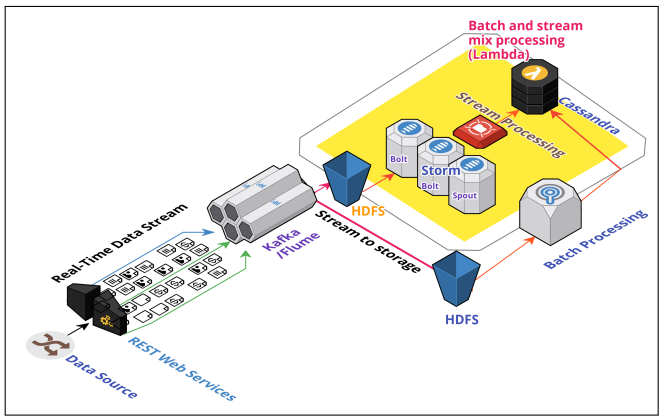

Fig. 1: Lambda Architecture integrates low latency real-time framework with high throughput Hadoop batch framework. Notice, data from Kafka message queue gets ingested to both and stream processor frameworks. While stream processors can analyze data in transit, batch module stores the ingested data pool into HDFS over the time before processing. Apache Storm and Hadoop MapReduce frameworks are used at stream and batch modules respectively. A NoSql datastore (Cassandra) combines the batch and real-time views at the serving layer.

B. Multi-agent Lambda Architecture

Multi-agent Lambda Architecture (MALA) is our extension to the standard LA model and primary contribution in this paper. MALA enables accumulative and collaborative learning through set of big data tools and methods. Stream and batch components act as two autonomous multi-agent system which collaborates to produce more comprehensive view of the data.

This research was supported by Accenture Technology Labs Beijing, China. Grant number RDF 15-02-35.

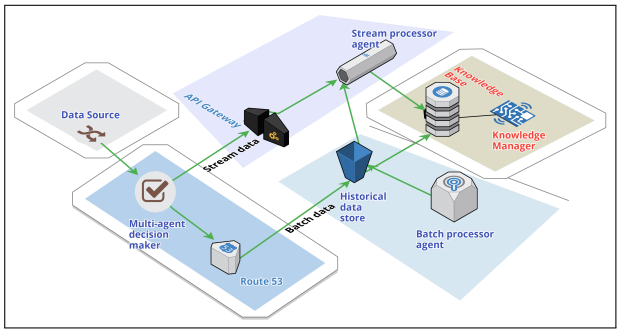

Fig. 2: Multi-agent Lambda Architecture (MALA). Modules of MALA are: Stream processing engine, historical data miner, Knowledge base and Knowledge miner. Batch and streaming modules are considered as two autonomous Multi-agent systems to collaborate and achieve a common goal.

1) Stream Processing Engine: Streaming processing engine initializes itself with trained model generated from batch stored into HDFS. Stream engine uses the batch data as initial offset to start with and builds on top of it, incrementally at a specified window intervals. The main challenge streaming layer faces is to process in-flight high velocity of ingested data without first storing into a file system or a database. In the current supporting case study, the stream engine ingests users clickstream data generated a high rate. Since volume of *click to view* is much higher than final purchases, the framework uses multi-node multi-broker Kafka installation for distributed caching and integrating pipeline [2]. Apache Storm is the subscriber for the Kafka message queues and does the distributed processing through series of Spouts and Bolts [3] [4] [5].

2) Historical Data Miner: Historical Data Miner(HDM) has two sub components: distributed storage and processing by MapReduce. Hadoop Distributed File System (HDFS) or a NoSQL datastores like Cassandra, MongoDB or HBase can be a storage option for batch jobs. Batch job uses Hadoop MapReduce through Java and Hive with a one-shot batch run on the entire data pool.

3) Knowledge Miner and Knowledge Base: Knowledge Minder (KM) is the severing layer to the end user. It combines the views from batch and stream to provide a lifelong learning mechanism. KM persists the results into the Knowledge Base (KB). KM also performs data filtration, provides the system health and performance statistics for data governance and monitoring purposes.

III. Implementation of a Recommender System Through Lifelong Learning Model

Incremental lifelong learning approach outlined through MALA architecture is implemented through a recommender system. Recommender system we design is capable of recommending items for an e-commerce site. It does not require users to login to see the list of recommendations. Rather, the system users click to view data as passive or implicit form of feedback [6] to recommend items. Implicit Feedback Based Recommender System (IFBRS) uses the concept of *Collaborative Filtering* [7] [8] [9] [10] to build upon other users viewing preference on the same items under one browsing session. The recommender system also considers filtration on item similarity based on categories.

Algorithm 1 Computing Recommendation Through MALA Architecture

Input: batch dataset d_b, streaming window dataset $d_s = \sum_{i=1}^{n} d_i$, Kafka topic k_t, Database address d_a,

Output: : Ordered recommendation list

1: $C_b \leftarrow$ CollaborativeFilteringFunction(d_b)
2: $S_b \leftarrow$ ItemSimilarityFunction(d_b)
3: call subscribeKafkaQueue();
4: **for** Streamed dataset d_s **do**
5: Stream engine consumes from Kafka queue
6: $C_s \leftarrow$ CollaborativeFilteringRule(d_s)
7: $R_c \leftarrow$ Update(C_b, C_s) //update batch with stream
8: $S_s \leftarrow$ ItemSimilarityFunction(d_s)
9: $R_s \leftarrow$ Update(S_b, S_s) //update batch with stream
10: R $\leftarrow R_c \cap R_s$
11: **return** R persisted into d_a
12: **end for**

Step 1 (lines 1-2): Initial recommendation list is built based on item similarity and collaborative filtering rule .
Step 2(lines 7-9): On each streaming window, batch data (plus previous iterations of streaming data) gets updated with current stream window
Step 3 (lines 10-11): Knowledge base is updated and used for next iterations. Notice, final recommendation is based on items common between Collaborative Filtration and item similarity rules.

A. Experimental Results

IFBRS is verified against Spark ALS for accuracy. Users clickstream data provides several latent factors like browsing location and time of browsing [11] [12]. IFBRS performs a weighted average with different latent factors to compute the final recommendations. Refer [13] for the detailed steps for core recommender engine.

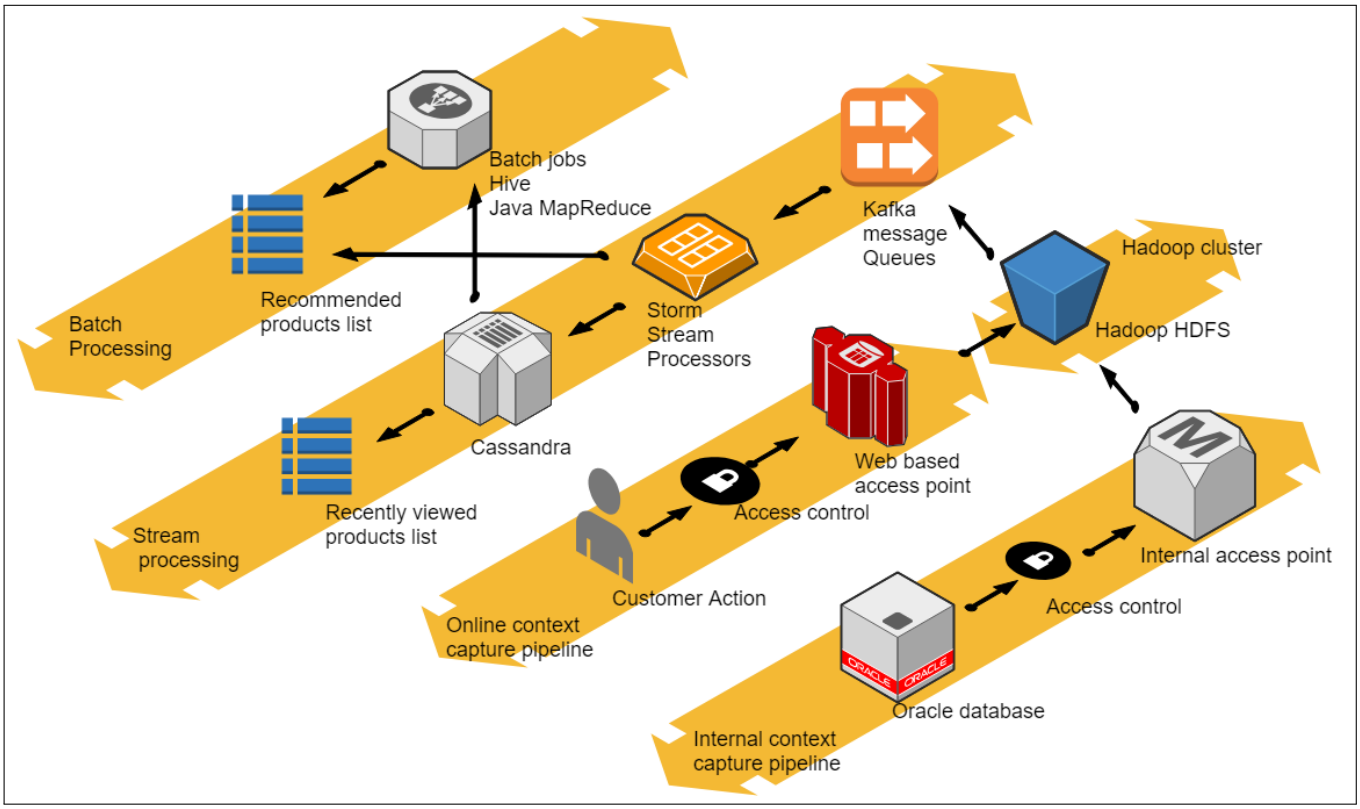

Fig. 3: Recommender system architecture and data flow. Kafka ingestion pipeline retrieves users click data. Apache Storm is the real-time processing engine. Internal context holds the items feature for similarity computation, online context captures the users clickstream data. Fusion of batch and real-time data flow updates the recommendation ordering making it more precise through every iteration.

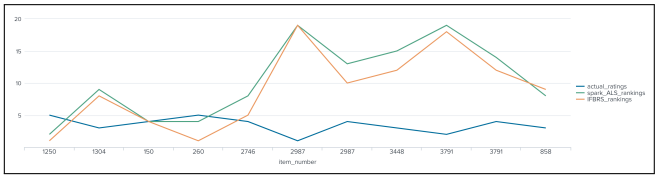

Fig. 4: IFBRS provides better accuracy than Spark ALS as its distribution over different product IDs (x axis) is closer to user provided actual ratings.

References

[1] J. Heidrich, A. Trendowicz, and C. Ebert, "Exploiting big data's benefits," *IEEE Software*, vol. 33, no. 4, pp. 111–116, 2016.

[2] G. Pal, G. Li, and K. Atkinson, "Big data real time ingestion and machine learning," in *2018 IEEE Second International Conference on Data Stream Mining Processing (DSMP)*, Aug 2018, pp. 25–31.

[3] D. Xiang, Y. Wu, P. Shang, J. Jiang, J. Wu, and K. Yu, "Rb-storm: Resource balance scheduling in apache storm," in *2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, Conference Proceedings, pp. 419–423.

[4] M. R. H. Farahabady, H. R. D. Samani, Y. Wang, A. Y. Zomaya, and Z. Tari, "A qos-aware controller for apache storm," in *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, Oct 2016, pp. 334–342.

[5] J. Xu, R. Rahmatizadeh, L. Bölöni, and D. Turgut, "Real-time prediction of taxi demand using recurrent neural networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 8, pp. 2572–2581, Aug 2018.

[6] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *2008 Eighth IEEE International Conference on Data Mining*, Conference Proceedings, pp. 263–272.

[7] H. Kim, S. Madhvanath, and T. Sun, "Hybrid active learning for non-stationary streaming data with asynchronous labeling," in *2015 IEEE International Conference on Big Data (Big Data)*, Conference Proceedings, pp. 287–292.

[8] C. H. Lee and C. Y. Lin, "Implementation of lambda architecture: A restaurant recommender system over apache mesos," in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, Conference Proceedings, pp. 979–985.

[9] A. Batyuk and V. Voityshyn, "Apache storm based on topology for real-time processing of streaming data from social networks," in *2016 IEEE*

First International Conference on Data Stream Mining & Processing (DSMP), 2018, Conference Proceedings, pp. 345–349.

[10] M. Hanif, H. Yoon, S. Jang, and C. Lee, "An adaptive sla-based data flow mechanism for stream processing engines," in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, Conference Proceedings, pp. 81–86.

[11] C. Xia, X. Jiang, L. Sen, L. Zhaobo, and Y. Zhang, "Dynamic item-based recommendation algorithm with time decay," in *2010 Sixth International Conference on Natural Computation*, vol. 1, Conference Proceedings, pp. 242–247.

[12] J. A. Deri, F. Franchetti, and J. M. F. Moura, "Big data computation of taxi movement in new york city," in *2016 IEEE International Conference on Big Data (Big Data)*, Dec 2016, pp. 2616–2625.

[13] G. Pal, G. Li, and K. Atkinson, "Multi-agent item to item contextual big data recommender system," *International Journal of Design, Analysis & Tools for Integrated Circuits & Systems*, vol. 6, no. 1, pp. p58–59, Oct, 2017.