Migrating a Recommendation System to Cloud Using ML Workflow

Dheeraj Chahal, Ravi Ojha, Sharod Roy Choudhury, Manoj Nambiar
TCS Research, Mumbai, India
{d.chahal, ravi.ojha, sharod.rchoudhury, m.nambiar}@tcs.com

ABSTRACT

Inference is the production stage of machine learning workflow in which a trained model is used to infer or predict with real world data. A recommendation system improves customer experience by displaying most relevant items based on historical behavior of a customer. Machine learning models built for recommendation systems are deployed either on-premise or migrated to a cloud for inference in real time or batch. A recommendation system should be cost effective while honoring service level agreements (SLAs).

In this work we discuss on-premise implementation of our recommendation system called iPrescribe. We show a methodology to migrate on-premise implementation of recommendation system to a cloud using ML workflow. We also present our study on performance of recommendation system model when deployed on different types of virtual instances.

CCS CONCEPTS

• **General and reference** → **Performance**; • **Computer systems organization** → **Cloud computing**;

KEYWORDS

Recommendation system, ML workflow, AWS SageMaker, cloud performance

ACM Reference Format:
Dheeraj Chahal, Ravi Ojha, Sharod Roy Choudhury, Manoj Nambiar. 2020. Migrating a Recommendation System to Cloud Using ML Workflow. In *ACM/SPEC International Conference on Performance Engineering Companion (ICPE '20 Companion), April 20–24, 2020, Edmonton, AB, Canada.* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3375555.3384423

1 INTRODUCTION

Recommendation systems are used in a wide variety of domains including retail, hospitality, media, video on demand etc. Organizations and businesses are using recommendation systems as vital means to improve revenue. Amazon's 35% sales and Netflix's 65% movie views are generated from suggestions given by their recommendation systems [11].

To improve the user experience, many portals use recommendation system that provides a list of products to the customer based on past history, user interests and his current requirements. The recommended products are shown to the user as the user logs in or starts browsing for some product. Some of the challenges in the recommendation system that affect the user experience are throughput, latency, accuracy (provide relevant products) etc. As observed, average user span for 17% web pages is less than 4 seconds [10]. Hence it is extremely important for medium and large size businesses to provide a low latency and high throughput recommendation system to convert new users to prospects. Moreover. prediction latency shall be bounded by tail latency to meet service level agreement (SLA) [15].

Developing and building models for recommendation system are of paramount interest to practitioners but equally important is how they are deployed and inference ready for optimal performance. Each of the performance attributes is affected by the algorithms or APIs used as well as the underlying hardware used.

In order to ease the work of model building, deployment and inference many organizations rely on ML workflows such as AWS SageMaker [1],Microsoft Azure ML [2] , TFX [13], MLFlow [16] etc. Many of these workflows are open source and allow users to build their solutions using either default API implementations or integrate their APIs and models with the workflow. Organizations aim to reduce the cost of their solution implementation by choosing the most appropriate hardware for model training, deployment and inference.

Once a model is deployed, inference is performed in real time or using batch processing. In real time, stream of incoming data is read as input and passed to the model for inference. Batch processing works on data stored in files and predictions are performed offline. A key challenge in both scenarios is to choose appropriate provisioning option among a large configuration space with diverse cost model. The chosen instance or cloud service should provide low-latency and cost-effective inference.

In this work we first discuss on-premise deployment of our previously developed content based recommendation system called iPrescribe [14]. iPrescribe extracts features about user profiles and products and makes recommendations for new items. Next, we discuss migration of this recommendation system to AWS instances using AWS SageMaker. We also compare the performance of recommendation system under different deployment schemes. More specifically, the contributions of our work are as follows:

(1) Migration methodology for a recommendation system to migrate from on-premise deployment to cloud using ML workflow.
(2) Recommendations for performance and cost-effective deployment of recommendation system.

ICPE '20 Companion, April 20–24, 2020, Edmonton, AB, Canada

ACM ISBN 978-1-4503-7109-4/20/04…$15.00
https://doi.org/10.1145/3375555.3384423

[1] https://aws.amazon.com/sagemaker/
[2] https://azure.microsoft.com/en-in/services/machine-learning/

Rest of the paper is structured as follows. In section 2 we discuss state of the art and related work. Our recommendation system is briefly discussed in section section 3 followed by approach to migrate the recommendation system to cloud in section 4. We discuss recommendation engine and inference in section 5. Experimental setup and evaluation is discussed in sections 6 and 7 respectively followed by conclusion in section 8.

2 RELATED WORK

Many organizations are adopting artificial intelligence and machine learning techniques for customer behavior studies. Lot of research has been done for building and deployment of recommendation systems. Uber has its own machine learning workflow called Michelangelo [6]. Michelangelo allows Uber to build, deploy and operate machine learning solutions at a very large scale. It has the capability to perform 1 million predictions per second. FBLearn [12], a ML platform used by Facebook engineers can deliver 6 millions predictions per second. Although recommendation engines have similar features but they can be deployed in different ways for inference in real time. Aloha [9], a model representation framework that mitigates the data related issues by allowing user to implement user-defined datatypes instead of framework defined datatypes. Crankshaw*et al.* [8] developed a low latency prediction serving system called clipper. Clipper demonstrates a low and bounded latency of less than 20ms. Google has developed an industrial grade prediction serving system [5] using TensorFlow training framework. Zhang *et al.* [17] recently presented their study on cost-effective and service level objective (SLO).

Although all these systems and workflows are able to deliver high throughput and low latency but none of these provide sufficient insight into cost-effective deployment. These require strategy from user to achieve high performance and cost-effective deployment. The onus is on user to decide the type of instance appropriate for model deployment and inference such that it is performance efficient and cost-effective as well.

Inference is an important component to explore further in a recommendation system and we believe that our work in this direction will provide insight to the practitioners for migrating on-premise deployment of ML models for inference to cloud.

3 OUR RECOMMENDER SYSTEM

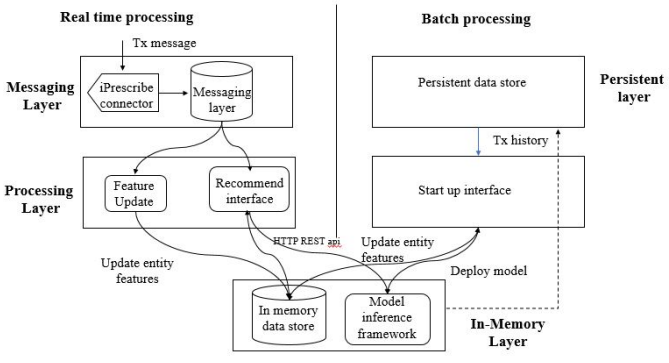


Figure 1: Abstract view of iPrescribe interfaces

iPrescribe is a customer-based predictive analysis recommender system to deliver variety of customer prepositions. The recommendation system model predicts the likelihood that the user will click on an advertisement. iPrescribe has two interfaces : Batch and Real time. iPrescribe uses XGBoost [7] ML algorithm which requires extraction of features from transaction data containing temporal behavior of users to build probability prediction model. The user features are updated continuously via real time interface. iPrescribe achieved significant accuracy as indicated by area under curve (AUC) value of .67 and F-score (Harmonic mean of precision and recall) value .348 [14]. In a real time deployment of iPrescribe, the user requests in B2C system are processed and inference is performed using the already built and deployed model. Model building in iPrescribe requires using features generated from the temporal behavior of the customer.

4 OUR APPROACH

In this section we first discuss on-premise deployment of iPrescribe and then its migration to AWS cloud using ML workflow called SageMaker. Both these deployment strategies and iPrescribe components are discussed as below:

4.1 On-premise iPrescribe

An abstract view of on-premise implementation of iPrescribe is as shown in the Figure 1. It has a batch interface called *Startup* and two real time interfaces namely *Recommend* and *FeatureUpdate*. The Startup interface processes transactional data for feature creation, training and testing the model. The concatenated feature one-hot vectors generated from feature creation process are used to build XGBoost model. Both recommend and FeatureUpdate are triggered via iPrescribe connector for every action message which need to generate an offer for a user. The recommend interface extracts the user context from the incoming message, fetched its feature for the model, builds concatenated one-hot vector and inference from the model to get the best offer for the user. The featureupdate interface is responsible for updating the features for every action performed. The process involves extracting the user details from the incoming message, retrieving the existing features, updating the features and saving it back in the feature store. Further details about on-premise implementation of iPrescribe are available in our previous work [14].

4.2 Cloud based implementaion of iPrescribe

Now we implement iPrescribe using MXNet [3] with SageMaker python SDK as shown in Figure 2. Our MXNet training scripts implement mandatory *train* function containing iPrescribe model training code. The *startup* interface discussed in on-premise implementation is implemented in this method. Train method is called by invoking *fit* method using MXNet estimator. The input to *fit* function is location of training data either stored locally or on S3 storage.

Model saving is enabled by *save* function to store and manage the model in S3 returned by *train*. After calling fit, we invoke *deploy* method using MXNet estimator to create a SageMaker endpoint. We specify instance type and count as arguments to the *deploy* function. *deploy* returns a predictor object that we use to inference

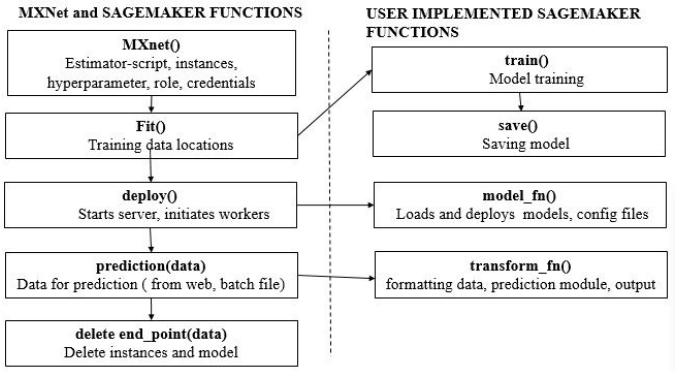


Figure 2: AWS implementation of iPrescribe using MXNet

on endpoint hosting our MXNet model. Each predictor contains a *predict* function that takes user request in the JSON format and returns the result of inference on the model. *predict* function invokes *transform_fn* to inference using deployed model. One hot vector generation and *predict_proba* function for inference are implemented inside *transform_fn*. The recommend interface from on-premise implementation is implemented in *transform_fn*. SageMaker provides default implementation of *train, save, model_fn and transform_fn* but we have modified these methods to contain the functionalities of iPrescribe.

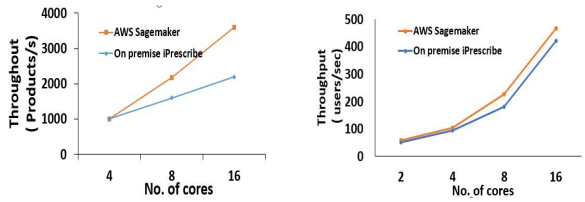
5 RECOMMENDATION ENGINE AND INFERENCE

In this section we discuss recommendation engine and inference in iPrescribe. The performance of both real time and batch interfaces depends upon feature store access time, processing time and inference time . Our focus in this work is performance of inference engine. iPrescribe prediction is done on-premise using python web server implementation. For our experiments we first train the model and deploy locally on our servers for inference. To simulate the behavior of user clicks we generate a stream of user records by starting multiple processes called producers that read input data from multiple files. Each data file contains records from different users sorted on clock time. We use HAProxy, an external load balancer for load balancing the data across multiple cores. HAProxy provides a fast and efficient load balancing for TCP and HTTP based applications. HAProxy uses round robin distribution policy and distributes data over various output ports where multiple consumers are listening for inference using deployed model. In order to avoid over subscribing or under subscribing the cores we keep number of producers and consumers equal to number of cores used for a run.

Next, we deploy iPrescribe on AWS cloud using SageMaker as discussed in section 4.2. For batch inference we read data for inference from multiple files saved in JSON format. In a batch inference, SageMaker reads the data from input files in mini batches. Input data for inference is read from a file stored on simple storage service (S3) and results are also stored back in S3. The model is deployed using HTTPs endpoint that can also be invoked in real time for inference. SageMaker uses Nginx [4] for load balancing.

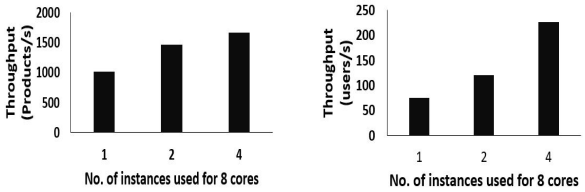
6 EXPERIMENTAL SETUP

We used well known benchmarks instacart [1]and PAKDD [2] to conduct experiments with on- premise and AWS deployment of iPrescribe. The on-premise server used for deploying iPrescribe had 28 physical cores. Instacart consists of 3 million grocery orders obtained from 200,000 instacart users. The data set consists of relation sets describing user purchase pattern over a period of time. We used *c-family* (compute optimized) and *m-family* (general purpose) instances of AWS for cloud based experiments. We control the ingestion rate by varying the number of producers. In our experiments throughput for instacart is measured as number of products predicted per second for a user while throughput for PAKDD is measured as prediction for number of users per second.



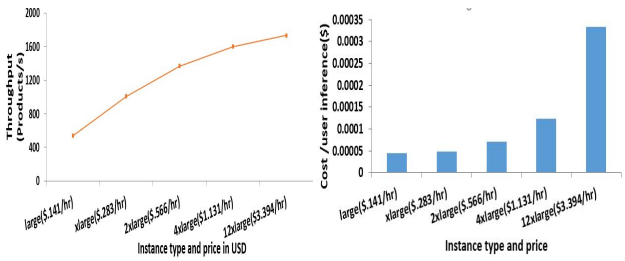
(a) (b)

Figure 3: On-premise and AWS SageMaker throughput (a) instacart (b) PAKDD



(a) (b)

Figure 4: Effect of number of instances on throughput (a) Instacart (b) PAKDD



(a) (b)

Figure 5: Instacart benchmark (a)Throughput on difference instance types (b) Cost per inference on different instance types

7 EXPERIMENTAL EVALUATION

We conducted several experiments to compare on-premise and cloud implementation of recommendation system. We also studied

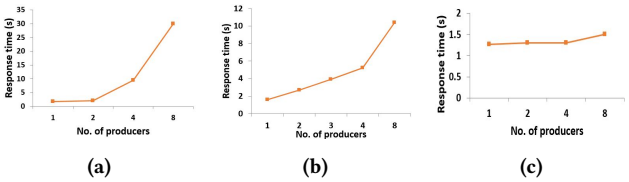


Figure 6: Response time of recommendation system using instacart benchmark on three different instances with (a) low network band width (5Gb/s) (b) low to mid network bandwidth (10 Gb/s) (c) High network bandwidth (25Gb/s)

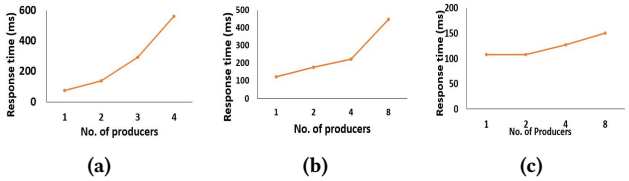


Figure 7: Response time of recommendation system using PAKDD benchmark on three different instances with (a) low network band width (5Gb/s) (b) low to mid network bandwidth 10 Gb/s) (c) High network bandwidth (25Gb/s)

the effect of instance configuration on performance and cost of recommendation system. In this section we present results of some the experiments.

Figures 3a and 3b show throughput for on-premise and sagemaker based implementation of iPrescribe using instacart and PAKDD benchmarks respectively in batch mode. We observed better throughput with AWS SageMaker which further improved as the number of active cores and workload is increased. Higher throughput is attributed to better optimizations and scaling available in the SageMaker.

In our next experiment we used m-family instances of AWS sagemake. We first ran our benchmarks on one ml.m5.2xlarge (8 cores) followed by two instances of ml.m5.xlarge (4 cores each) instance and lastly four ml.m5.large (2 cores each) instances. In these three runs total number of cores used (8 cores) and total cost (USD) is same but number and type of instances is different. As shown in Figure 4, throughput of the recommendation system increases as we increase the number of instances while keeping the total number of cores same. Maximum throughput was observed with 4 instances of ml.m5.large. This is due to the fact that size of mini batch in SageMaker that can be allocated to an instance at a time is limited by parallelism and size of records. Hence more instances result in more number of mini batches executed in parallel on instances available for inference.

Figures 5a and 5b show throughput and cost per inference of instacart benchmark on five SageMaker instances differing in number of cores, available memory and their usage cost. As shown in Figure 5b, cost per inference increases significantly by using an instance with large number of cores and higher costs but relatively smaller increase in the throughput is observed for the same instance as shown in Figure 5a. Hence prompting for a trade-off in cost and throughput.

Figures 6 and 7 show the effect of available network bandwidth on response time of benchmarks in real time. We used three types of AWS instances with different network bandwidths. We see a significant improvement in the response time by choosing instances with larger network bandwidth. For a high available network bandwidth we see that response time is almost constant even with increasing workload. We can conclude that not only the cores available but also network bandwidth affect the response time of the recommendation system.

8 CONCLUSION

We have successfully migrated an in-house recommendation system called iPrescribe from on-premise deployment to AWS cloud using automated ML workflow called SageMaker. In this work, we have studied performance metrics of iPrescribe including throughput, response time and cost per inference. Based on experimental results, we can conclude that by using cloud resources judiciously and appropriately, we can achieve our objective of better performance and cost-effective deployment of a recommendation system.

REFERENCES

[1] 2017. KaggleInstacartChallenge. https://www.kaggle.com/c/instacart-marketbasket-analysis.
[2] 2017. KaggleInstacartChallenge. http://www.recobell.com/rb/main.php?menu=pakdd2017..
[3] https://mxnet.apache.org/, 2019. [Online;Accessed 01 January 2019].
[4] nginx[enginex].http://nginx.org/en/, 2019. [Online;Accessed 25 December2019].
[5] Tensorflow serving. https://www.tensorflow.org/, 2019. [Online;Accessed 20 January 2019].
[6] Uber michelangelo. https://eng.uber.com/michelangelo/, 2019. [Online;Accessed 20 January 2019].
[7] XGBoost - Extreme Gradient Boosting. https://xgboost.readthedocs.io/en/latest/tutorials/model.html, 2019. [Online;Accessed 25 December2019].
[8] Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., and Stoica, I. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, Mar. 2017), USENIX Association, pp. 613–627.
[9] Deak, R. M., and Morra, J. H. Aloha: A machine learning framework for engineers. In *Proceedings of the SysML Conference* (2018).
[10] Gaille, B. Average Web Page Load Times By Industry. http://www.byreputation.com/Average-Web-Page-Load-Times_a/452.htm, 2015. [Online;Accessed 28 June 2015].
[11] Ge, W. A continuous dataflow pipeline for low latency recommendations. Master's thesis, KTH, School of Information and Communication Technology (ICT), 2016.
[12] Hazelwood, K., Bird, S., Brooks, D., Chintala, S., Diril, U., Dzhulgakov, D., Fawzy, M., Jia, B., Jia, Y., Kalro, A., et al. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2018), IEEE, pp. 620–629.
[13] Modi, A. N., Koo, C. Y., Foo, C. Y., Mewald, C., Baylor, D. M., Breck, E., Cheng, H.-T., Wilkiewicz, J., Koc, L., Lew, L., Zinkevich, M. A., Wicke, M., Ispir, M., Polyzotis, N., Fiedel, N., Haykal, S. E., Whang, S., Roy, S., Ramesh, S., Jain, V., Zhang, X., and Haque, Z. Tfx: A tensorflow-based production-scale machine learning platform. In *KDD 2017* (2017).
[14] Singhal, R., Shroff, G., Kumar, M., Choudhury, S. R., Kadarkar, S., Virk, R., Verma, S., and Tewari, V. Fast online'next best offers' using deep learning. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data* (2019), ACM, pp. 217–223.
[15] Yun, J.-M., He, Y., Elnikety, S., and Ren, S. Optimal aggregation policy for reducing tail latency of web search. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2015), SIGIR '15, Association for Computing Machinery, p. 63–72.
[16] Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., et al. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull. 41*, 4 (2018), 39–45.
[17] Zhang, C., Yu, M., Wang, W., and Yan, F. Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)* (2019).