Designing ML-based Approximate Query Processing Services on Time-Varying Large Dataset for Distributed Systems

Kihyuk Nam, Sung-Soo Kim, Choon Seo Park, Taek Yong Nam, Taewhi Lee
ETRI
Daejeon, Korea
{nam, sungsoo, parkcs, tynam, taewhi}@etri.re.kr

Abstract— Approximate query processing (AQP) has been well established for big data analytics to complement performance degradation due to the ever-increasing size of datasets. The evolution of machine learning technologies creates another opportunity for improving the AQP. There are two architectural aspects that should be considered for AQP-based data analytics services to embrace the trends. First, the services should support rapidly changing data. Second, the systems should manage the life-cycle of the machine learning process and accommodate the diversity of ML technologies. This paper discusses the requirements and design considerations for ML-based AQP services on time-varying large datasets for distributed environments in a system-independent way.

Keywords—Approximate Query Processing, AQP, Machine Learning

I. Introduction

Exact matching in queries for rapidly increasing datasets often shows poor performance. Approximate query processing (AQP) has emerged to remedy the problem by fast producing statistically equivalent results, and it has become a well-established component of big data analytics. Furthermore, recent development of machine learning (ML) technologies opened new opportunities in this direction. Many researchers have been suggesting a variety of ML-based methods to surpass the performance of traditional statistical methods.

AQP systems can roughly be classified into sampling and synopsis, which can be based on online or offline. Online-based systems dynamically take samples during querying, while offline-based systems use synopses made in advance. While ML-based AQP can naturally be categorized into synopsis by treating learned models as data summaries, the model can be different from the traditional table structure so that some kind of transformation may be required. The architecture and process can also reflect the nature of ML processes. Furthermore, the pace of ML development and data increase is so rapid that it should be designed to embrace heterogeneous methodologies and support continuous or periodic retraining models efficiently. The changes in deployment environments should also be considered as well. Data systems such as DBMS, big data systems, and EDA tools are moving into cloud environments to maximize the extensibility and maintainability. The AQP subsystems should support such environmental changes by designing distributed and system-independent architecture.

This paper defines core requirements for the state of the art, ML-based AQP services, then discusses design issues, and finally suggests possible solutions by specifying a baseline architecture, execution flows, integration with DevOps/MLOps, and interfaces including remote/local APIs and model packaging.

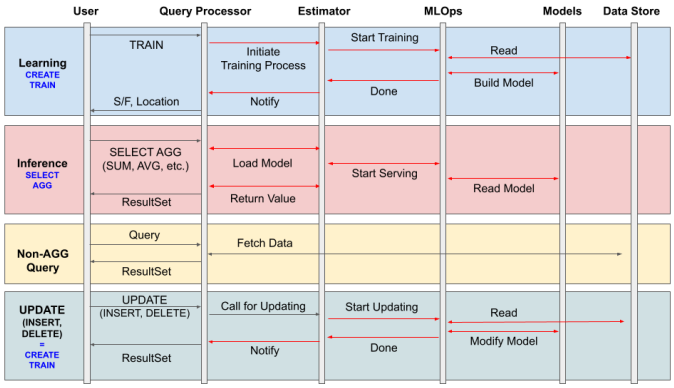

Fig. 1. Execution Flows

II. Requirements

This paper focuses on the ML-based AQP services on time-varying large datasets for deploying distributed cloud environments. Their requirements can be categorized into users, system properties, and usage scenarios.

A. Users

- Data analysts or their tools such as EDA, DBMS
- IoT devices with restricted resources
- Managers who add/modify/remove models and data

B. System Properties

The system must satisfy the following properties:

- Fast response time (learning and inference)
- Provides accuracy info such as confidence intervals
- Synopses management including automatically retraining models reactively or periodically, deploying models, and versioning their instances and types
- Collects and utilizes query histories for maximizing the quality and performance of the system
- Interfaces for distributed environments: a driver/local library or an API for remote access (REST)

C. Usage Scenarios

The use cases can be further divided as follows (Fig. 1):

- **Learning**: users choose the type of model and create their instances by initiating a learning process for the specified data. The corresponding SQL command is CREATE. The system should provide a list of supported types of models (e.g., RSPN, GAN, etc.).

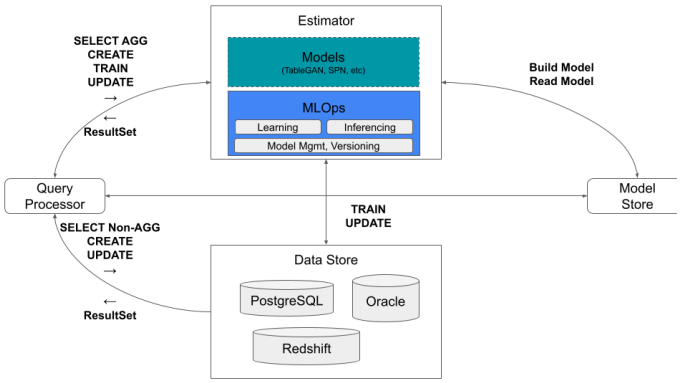

Fig. 2. System Design

TABLE I. API SPECIFICATION

Name			
train	Function	Initiates learning process with given parameters.	
	Parameters	*model*	Type (and instance) of the model to be trained. (e.g., SPN, TableGAN)
		data	The name and the location of the learning data. (e.g., Instacart, orders.csv) The data can be in csv, or DBMS.
	Returns	URI or ID of the learned model	
estimate	Function	Computes the approximate value for the given query	
	Parameters	model	Trained model to be queried
		query	SQL commands with aggregations. (e.g., SELECT COUNT(*) FROM orders where order_dow = 3). Default aggregate operations are COUNT, SUM, and AVG.
		data	Table name (e.g., FROM clause of SQL statements)
	Returns	A double (floating-point) value with a confidential interval (if available)	
check	Function	Checks if the training is finished	
	Parameters	uri	URI of the model to be learned
	Returns	A Boolean value	

- **Inference**: the service answers queries by computing approximate aggregation from the learned models. The corresponding commands are SELECT with aggregations (e.g., count, sum, average).
- **Update**: changes in the dataset can trigger the modification of the model to keep it up-to-date. It can be asynchronous when the underlying data storage is updated independently, or synchronous when the user inputs commands such as INSERT, DELETE, and UPDATE. It spans the learning process and the data store. Implementation can utilize the components from both subsystems.

III. System Overview

The system can be organized into four components: query processor, estimator, model store, and data store (Fig. 2). The query processor parses user inputs and sends them to the estimator if they contain aggregations (e.g., SELECT COUNT/SUM/AVG), or training commands (e.g., CREATE, TRAIN), or update commands (e.g., UPDATE, INSERT and DELETE). Otherwise, it talks directly to the data store. For a query processor, the estimator can be seen as a data store since it returns result sets with approximations from learned models. Depending on the type of commands, it can initiate learning processes or model serving processes. It can also provide a list of model types and instances that are currently available.

The estimator orchestrates ML operations. Models can be in two forms: types and instances. Types define the structure and methods of models, while instances are results from learning processes with specific versions of dataset. The users should register the types they want in advance. The same type may produce different instances if the datasets are modified or hyperparameters are changed.

The estimator delegates the detailed operations of ML processes to the MLOps subsystem. The system manager can build it from scratch by customizing open-source products such as KubeFlow [9] or can use cloud-based outsourcing services.

The data store can be any system that provides storage functionalities. It can be traditional DBMS, filesystems, or cloud services. The system should provide store independence in such a way that any store can be added or removed without affecting other subsystems. AQP systems can also be a store if they are coupled with a specific data storage (e.g., BlinkDB [16]). It's the query processor's job to coordinate the execution flows between the estimator and the specific data store.

The model store keeps all types and instances of data models. The actual binaries can be stored in the data store, but the control endpoint should be separated from the model store. The key role is the versioning instances and types.

The most important thing in this architecture is the method of interaction between subsystems. This paper assumes distributed, system-independent environments, so that a standardized remote API should be provided on top of the major operations. It can optionally provide a library that can be imported from the subsystems locally. Table I and Fig 3 show a REST API specification for the system and its proof-of-concept implementation.

IV. Design Considerations

There are some issues that should be considered during the design stage.

A. Managing Heterogeneous Models

New methods are emerging every year especially due to the rapid growth of the machine learning community. Our system aims to provide state-of-the-art AQP functionalities and to embrace heterogeneous models from different technologies, there should be an interface and packaging mechanism to support them. There are three types of contact points between the core systems and the added models.

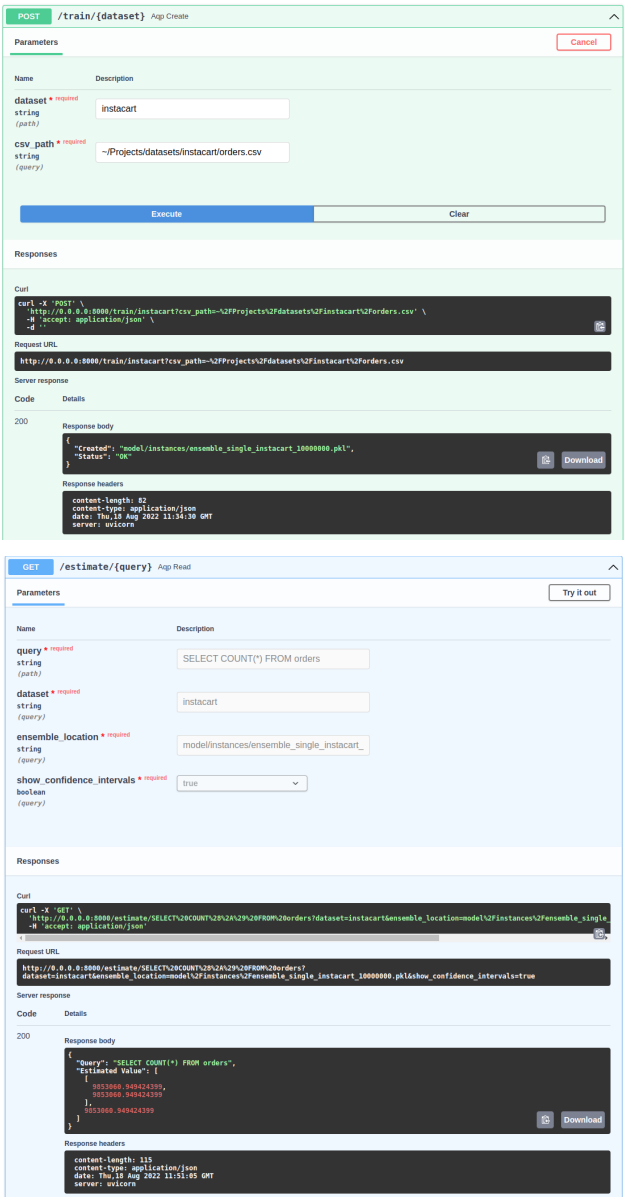

Fig. 3. REST API Prototype

- **Model types** define the structure and operations for learning and inference processes. For example, the system could provide models based on RSPN [3] and TableGAN [4]. The underlying mechanisms are different but their interfaces should be the same for the system to understand how to control them. For example, the traditional object-oriented technique can be applied by defining a common abstract class. The model providers would derive their own mechanism from it. While some manual efforts are required, it's a well-established and reliable way.
- **Model instances** mean generated binaries from the learning process. For example, you can support the ONNX [18] or choose a specific ML framework such as PyTorch [21]. Or you can go to a lower level by defining the learning and inference mechanism using native formats such as '.pkl' (Python's serialization format).
- **Model execution** takes charge of running appropriate models for given queries. It should be aware of how to load and call the models during runtime. For example, the system should choose an appropriate method to process the given 'SELECT' queries. The list of the supported operations can be provided by the model providers as a query method or standard formats such as JSON [17].

One may go further by adopting container technologies such as Docker [16]. The advantage of this approach is that the system can be platform-agnostic. Just packaging all the required components in a container would be enough while any duplication between containers or their layers may degrade the performance.

One of the key points for managing heterogeneous models is the versioning mechanism between model types, model instances, and their corresponding datasets. From a simple hash to a 'git'-like system can be used only if they guarantee the consistency between them.

B. REST API vs Message Queue

One of the popular interfaces for distributed systems is the HTTP-based REST API. However, there are a few issues that must be considered before implementation. Firstly, the training stage is by nature asynchronous. It usually takes a long time for very large datasets to finish learning. It will or must be finished but the exact time is unpredictable. Therefore, it may waste resources if the learning APIs synchronously wait for the process to finish their jobs. They must be able to selectively handle the learning tasks synchronously or asynchronously. Secondly, the system should recognize the history of state changes of the processes. For example, there can be **i** users who send **j** queries (commands) to the system, which manages **k** states (at least three internal states: ready, running, error) for one of **l** operations. Therefore, the number of states(**i*j*k*l**) the system should manage may increase drastically. There are two ways to solve the issue.

- Use a message queue such as RabbitMQ [10]. It manages requests/responds consistently but both the estimator and its users would depend on the adopted solution.
- Adopt MLOps Frameworks such as KubeFlow [9] which provides REST interfaces with the consistent mechanism for managing models and processes.

C. Data-Driven vs Workload-Driven

There are a few ways of categorizing AQP techniques (e.g., Table II). Traditional AQPs used sampling, but there have been many attempts to apply ML techniques these days. The approximation can be done during runtime, which is called online methods [19], or use a summary information which may be in the form of tables or model instances, that is prepared in advance offline. Another angle that we can view the AQPs is whether the models are learned directly from data or indirectly from query history. The system designers should carefully analyze their requirements and environments and then decide the best combination of those aspects.

Instead of blindly adopting deep neural networks, efficient probabilistic graphical models such as Sum-Product Networks (SPN) [11] can be a good fit for data models since they can directly learn data, their learning and inference performance is competitive, and there is an open-source framework called SPFlow [1]. The columns(features) of the datasets can be represented as variables of the joint probabilistic distributions.

TABLE II. CLASSIFICATION

Name	Online	Method	Environment
BlinkDB [15]	Offline	Stratified sampling	Distributed
Verdict [19]	Online	Database Learning	Stand-alone
DeepDB [3]	Offline	Data-driven, SPN [11]	Both
DBEst++ [20]	Offline	Data-driven, MDN	Both

V. CONCLUSION AND FUTURE WORKS

This paper suggests a practical way of providing a system independent AQP service for various applications in a distributed environment. Based on the base architecture, one can easily use, extend and maintain AQP systems for continuously evolving large datasets. Connecting to ML-data management systems such as MindsDB [14], or middleware for AQP like VerdictDB [2] can increase its utilization.

Computing temporal aggregations for analyzing time-varying features of datasets such as TATS [5] can increase the applicability of the system. The ability to analyze data from a real-world environment using multimodal mobile sensors is also interesting. For example, the lifelog dataset [6] measured human behavior and status for a long time from IMS, GPS, microphone, temperature, humidity, air pollution, and wristband sensors such as PPG and EDA for physiological data. The fast-growing spatiotemporal data can also be a good target of our system.

ACKNOWLEDGMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2021-0-00231, Development of Approximate DBMS Query Technology to Facilitate Fast Query Processing for Exploratory Data Analysis)

REFERENCES

[1] https://github.com/SPFlow

[2] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. 2018. VerdictDB: Universalizing Approximate Query Processing. In Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18). Association for Computing Machinery, New York, NY, USA, 1461–1476.

[3] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: learn from data, not from queries! Proc. VLDB Endow. 13, 7 (March 2020), 992–1005

[4] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. 2018. Data synthesis based on generative adversarial networks. Proc. VLDB Endow. 11, 10 (June 2018), 1071–1083.

[5] Shin, Y.-O., Park, S.-K., Baik, D.-K. and Ryu, K.-H. (2000), TATS: an Efficient Technique for Computing Temporal Aggregates for Data Warehousing. ETRI Journal, 22: 41-51.

[6] Chung, S., Lim, J.M., Lim, J., Ju Noh, K., Kim, G., Jeong, H.. Real-world multimodal lifelog dataset for human behavior study, ETRI Journal 44(2022), 426– 437.

[7] MA, Qingzhi, et al. Learned Approximate Query Processing: Make it Light, Accurate and Fast. In: CIDR. 2021.

[8] Li, K., Li, G. Approximate Query Processing: What is New and Where to Go?. Data Sci. Eng. 3, 379–397 (2018).

[9] www.kubeflow.org

[10] www.rabbitmq.org

[11] H. Poon and P. Domingos, "Sum-product networks: a new deep architecture," in Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI), pp. 337–346, 2011.

[12] D. Lopez-Paz, P. Hennig, and B. Scholkopf, 2013. The randomized dependence coefficient. In Advances in neural information processing systems, 1–9.

[13] Gens, R., and Domingos, P. 2013. Learning the Structure of Sum-Product Networks. In Proc. of ICML.

[14] www.mindsdb.com

[15] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys '13). Association for Computing Machinery, New York, NY, USA, 29–42.

[16] www.docker.com

[17] www.json.org

[18] https://onnx.ai

[19] B. Mozafari. Verdict: A system for stochastic query planning. In CIDR, Biennial Conference on Innovative Data Systems, 2015.

[20] Qingzhi Ma, Ali Mohammadi Shanghooshabad, Mehrdad Almasi, Meghdad Kurmanji, Peter Triantafillou. Learned Approximate Query Processing: Make it Light, Accurate and Fast. In 11th Conference on Innovative Data Systems Research, CIDR 2021,

[21] www.pytorch.org