STRATUM: A BigData-as-a-Service for Lifecycle Management of IoT Analytics Applications

Anirban Bhattacharjee, Yogesh Barve, Shweta Khare,
Shunxing Bao, Zhuangwei Kang and Aniruddha Gokhale
EECS Dept, Vanderbilt University, Nashville, TN, USA
{anirban.bhattacharjee; yogesh.d.barve; shweta.p.khare;
shunxing.bao; zhuangwei.kang; a.gokhale}@Vanderbilt.Edu

Thomas Damiano
Lockheed Martin Advanced Technology Labs
Cherry Hill, NJ, USA
thomas.a.damiano@lmco.com

***Abstract*—Smart Internet of Things (IoT) applications require real-time and robust predictive analytics, which are based on Machine Learning (ML) models. Building ML models from Big Data is not only time-consuming, but developers often lack the needed expertise for feature engineering, parameter tuning, and model selection. The proliferation of ML libraries and frameworks, data ingestion tools, stream and batch processing engines, visualization techniques, and the range of available hardware platforms further exacerbates the system design, rapid development, and deployment problems. Finally, resource constraints of IoT require that the execution of the analytics engine be distributed across the cloud-edge spectrum. To overcome these daunting challenges, we present *Stratum*, which is an event-driven Big Data-as-a-Service offering for IoT analytics lifecycle management. Stratum provides users with an intuitive, declarative mechanism based on the principles of model-driven engineering to specify the application and infrastructure requirements. It automates the deployment via generative programming principles. This paper highlights the problems that Stratum resolves, demonstrating its capabilities using real-world case studies.**

***Index Terms*—AI/ML platform, Big Data Analytics, IoT, Cloud/Edge Resource Management, Deployment, Automation, Domain-Specific Modeling.**

I. Introduction

The *Internet of Things (IoT)*-based systems generate high volumes of data at high velocity, which must be analyzed to derive valuable insights and make informed decisions for a variety of application domains, e.g., video analytics, predictive analytics, recommendation systems rely on the live and in-depth analysis of the incoming data streams as well as the historical data [1]. In this context, the development, deployment, and execution lifecycle of IoT analytics tasks are significantly complicated. First, it entails developing one or more artificial intelligence (AI)/machine learning (ML) models using large training data sets. This step requires the developers to be cognizant of the range of feasible ML models (e.g., linear models, decision trees or deep neural networks), and be able to select from among the plethora of ML libraries and frameworks. Second, once the ML models are trained and ready for serving the incoming requests, they must be rapidly deployed and integrated with the analytics pipeline on the target hardware infrastructure.

Although the ML model development can be carried out in resource-rich clouds, the resource-constrained nature of IoT systems and the real-time requirements of the analytics tasks preclude the movement of large amounts of data from the edge to the cloud for prediction. Instead, effective resource management decisions are needed to partition and distribute the trained ML models across the cloud-edge spectrum [2].

Unfortunately, IoT analytics application developers often do not possess the expertise required in handling all the challenging lifecycle activities of IoT analytics. Thus, there is a compelling need to ease the ML model development process and relieve the developer from the responsibility of having to determine the placement of analytics application components, monitoring their resource usage, and controlling different data processing tasks across the cloud-edge spectrum [3].

Model-driven Engineering (MDE) [4], specifically domain-specific modeling languages (DSMLs) and generative programming [5] are known to provide intuitive abstractions to users relieving them from error-prone and repetitive tasks while serverless computing relieves the user from runtime infrastructure management issues. In this paper, we exploit the benefits of MDE and serverless computing, and present a Big Data-as-a-Service called *Stratum* for IoT analytics lifecycle management. Our preliminary work on Stratrum [6] presented only the vision; this paper delves into the design and evaluates Stratum's capabilities using real-world use cases.

Organization of the Paper: The rest of the paper is organized as follows: Section II presents a survey of existing solutions in the literature and compares them to Stratum; Section III presents the background and problem formulation; Section IV presents the design of Stratum; Section V evaluates Stratum in the context of a prototypical case study; and finally, Section VI presents concluding remarks.

II. Related Work

In this work, we compare and contrast Stratum with relevant state of the art solutions for end-to-end lifecycle management of big data analytics tasks.

Ease.ml [7] is a training platform providing automatic model selection using a declarative programming approach. Ease.ml introduces a resource scheduler to manage the deployment of the training job in a shared cluster environment used by multiple users at once. Similarly, TFX [8] is another ML platform at Google, which provides tools for data preparation

and model inference serving. Michelangelo [9] is an ML-as-a-service framework deployed at Uber that facilitates building and deploying ML models in the cluster environment. Their DSL enables users to define the ML tasks to be used for both training and inference ML jobs. However, the DSL restricts the users from choosing only the algorithms that are supported by the platform, thus limiting users from experimenting using different ML algorithms and frameworks.

To the best of our knowledge, Michelangelo, and Alchemist are proprietary tools Commercial services such as Amazon SageMaker, Microsoft Azure service, and IBM's Watson Studio support both ML training and deployment pipelines, but are constrained to their underlying proprietary runtime infrastructures, which can potentially result in vendor lock-in. In contrast, Stratum framework offers the user to choose their ML framework generating ML code for those runtimes. Moreover, the addition of new ML libraries and frameworks are relatively straight-forward in Stratum.

MLFlow [10] is an Python based open source ML platform project that covers end to end lifecycle phases of ML development and deployment. ML.NET [11] is an open source ML pipeline framework by Microsoft. ML models can be integrated directly into application codebase natively. Other efforts such as Weka, Apache Mahout, Scikit-Learn provides declarative programming means to design and create ML pipelines and models. However, these platforms do not provide the means for model versioning and model deployment.

In comparison, Stratum provides a unified framework that supports design-time tools and deployment tools for model construction and deployment. It handles deployment across a heterogeneous set of platform spanning from cloud to edge computing platforms. Stratum also provides version support while creating designing models using a visual drag and drop GUI interface. Stratum leverages MDE technologies that facilitate creating custom DSMLs, automated code generation facility, and orchestration of models to be deployed on the target platforms. We believe these end-to-end capabilities are lacking in the current state of the art integrated tool suites which Stratum addresses.

III. Problem Formulation

In this section, we use a motivating scenario to highlight the challenges and derive the solution requirements for Stratum.

A. Motivating Case Study Eliciting Key Challenges

Consider an IoT use case of an automated toll booth which takes images of a vehicle's license plate to charge toll [12]. This application will involve an image recognition service comprising predictive analytics so that it can automatically detect the license plate of the entering or exiting car from a toll plaza. The overall system will thus involve a camera that takes a picture of the license plate, analyzes the image to identify the license plate and accordingly charge the associated account with the appropriate toll.

Challenge 1: Model Development: Building predictive analytics requires developing AI/ML learning models based on historical datasets, which is a challenging task and requires domain expertise. The developer is faced with a diverse set of ML capabilities including classification, regression, recommendation (ALS), clustering etc. that are provided by different libraries and frameworks, such as Scikit-learn, Spark MLlib, etc. Developing highly accurate models using feature engineering, model selection, hyperparameter tuning in those platforms is hard. Moreover, to speed up the training process, it needs to be driven by high-performance computing using GPU and CPU, and needs to be distributed, if possible. Unfortunately, all ML frameworks and algorithms do not support distributed model training. Finally, the model development platforms must be able to rapidly select the best models by evaluating a large number of models in a distributed manner.

Challenge 2: Model Deployment: For IoT applications, such as the toll booth use case or for automation assistants like Google Home that use natural language inference models, the device-to-cloud data round trip latency is considerable and hence processing at the edge is attractive as it reduces latency and makes connected applications more responsive. Thus, after the AI/ML model is trained, it needs to be rapidly pushed to IoT devices for big data inference. The cloud resources involve a variety of servers including energy efficient multi-core processors and GPGPUs (e.g., NVidia TX1), or it can be a private cloud with limited processing power. Thus, effective selection of resources is necessary.

Challenge 3: Data Movement and Management: Edge devices send the filtered data to the cloud using one of many communication protocols, e.g., HTTP, MQTT and data ingestion services such as Apache Kafka, Apache Nifi, or Amazon Kinesis must be programmed to listen to incoming data streams and be able to retain the data in databases or data lakes. Once the data is ingested in the server, different subscribers may be interested in the outcome so they can run separate live analytics on the window of streaming data and visualize the live patterns [6]. Developers are unlikely to be experts in all of these technologies and protocols to deploy and autoscale the complete data analytics pipeline.

Challenge 4: Determining the Right Hardware: Batch processing frameworks such as Apache Hadoop or Spark can run deep analytics by aggregating data from multiple data sources. These frameworks can integrate the trained ML model for their diagnostic or predictive analytics as required. The challenges lie in determining the right hardware to use from the wide variety of device classes, choosing deployment options (on-premise or over cloud), and configuring the deployment platform for the business application to deliver real value. Moreover, all the performance metrics of the hardware components need to be monitored for reactive and proactive decision making to auto-scale the application.

B. Solution Requirements

Based on the elicited challenges from Section III-A, we present the solution requirements that Stratum must meet.

Requirement 1: Flexible ML Service Development and Encapsulation: To address Challenges 1 and 4 , there is a need

for an AI/ML model building framework that can decouple the ML algorithms from existing ML frameworks and generate code for the underlying framework from the users' high-level ML model. The model should be encapsulated, containerized and exposed via REST APIs.

Requirement 2: Automated Deployment of Application Components in Heterogeneous Environment: To address Challenges 2, and 4, we need a capability that reduces developer effort. The developer should specify only minimal information using a self-service framework, and the system then automate the deployment and data movement challenges.

Requirement 3: Performance Monitoring and Intelligent Resource Allocation: To address Challenges 3, and 4, the solution must integrate data and storage services, schedule the workload on container deployments that are orchestrated by an execution engine with continuous monitoring of system metrics, and integrate a performance data collection strategy. Using these metrics, it must be able to allocate resources effectively for the specific tasks dynamically and proactively.

IV. Design and Implementation of Stratum

This section describes the design of Stratum and shows how it meets the requirements of Section III-B. Stratum is realized around the core concepts of Model-Driven Engineering (MDE) and generative programming. Stratum provides a graphical interface to compose and deploy the ML pipeline using higher-level intuitive abstractions. The abstract syntax and semantics of the DSML are captured concretely in multiple metamodels. The user-defined specifications are used to automate the entire ML workflow for the user. The Stratum DSML and its underlying metamodels, constraint checkers, and code generative capabilities are built on top of WebGME modeling environment [13].[1] Figure 1 illustrates the overall usage workflow of the Stratum model-driven framework to realize a big data IoT analytics application.

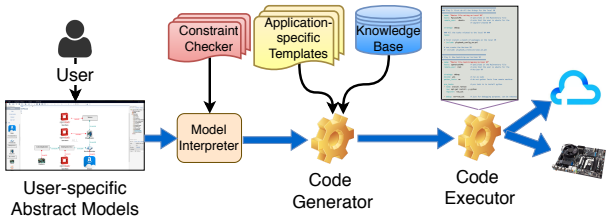

Fig. 1: Stratum Workflow to Realize an IoT Analytics Pipeline

A. Addressing Requirement 1: Development Kit for AI/ML Model Development

Stratum comprises a "Development Kit" for ML service development and encapsulation. It is an integrated model-driven environment aid to build ML pipelines by abstracting data preprocessing strategies, ML algorithms that execute on existing ML frameworks, and evaluation strategies.

[1]Due to space limitations, we do not show details of the metamodels.

1) Overview of the Development Kit: Stratum's development kit provides a standard data exchange format that supports read and write capabilities in different data formats from different data sources. It also provides unified data access across different machine learning frameworks such as Scikit-learn, SparkMLlib, and TensorFlow. The integrated ML pipeline allows retrieval (input), and manipulation of data stored in cloud storage such as Amazon S3, or Azure DataLake Store, or locally or in a scalable datastore (e.g., HDFS-based) via RESTful and other microservice interfaces (e.g., Data Preprocessing, Stream Analytics).

We integrated Jupyter Notebook and Apache Zeppelin (notebook-based environment) to provide data-scientists the ability to train their models interactively. The data-scientist can also directly generate Python code, if needed. The steps in modeling and generative capabilities of Stratum are shown in Figure 1. The Deployment Platform can be a distributed system or standalone hardware. The deployment of the machine learning pipeline on the target machine for training is handled by Stratum as described in Section IV-B.

2) Metamodel for Machine Learning Algorithms: We capture various facets of the application and target machine hardware specifications in the metamodel. We captured the specifications for a diverse set of ML algorithms including classification (e.g., logistic regression, naive Bayes), regression, decision trees, random forests, recommendation (ALS), clustering (K-means, GMMs), and many others in the Stratum metamodel. Using this metamodel, the data-scientist can drag relevant machine learning blocks, and define all the parameters such as fit_intercept, normalize, n_jobs for Scikit-learn linear regression block or specify the type of layers such as dense, CNN, RNN for deep learning.

3) Model Evaluation and Flexible ML Service Encapsulation: Users can express pipelines as a directed acyclic graph (DAG) where each node represents a task such as data preprocessing, training based on different ML techniques, and deployment to evaluate the model. The hyperparameter tuning methods such as Grid Search, Bayesian optimization, Gradient-based Optimization, Reinforcement Learning based optimization, etc are an advanced mechanism to evaluate the model, and this is provided as placeholders in Stratum.

The ML algorithms are encapsulated in Linux containers and exposed using endpoints. The DSML encapsulates the ML algorithms in Linux containers to support parallel execution of the pipeline based on the availability of the resources. After training the model, we evaluate the model based on different scoring methods such as accuracy, f1 score, precision, r2 score etc. based on the user's choice. Based on the evaluation criteria, the best ML model is selected and saved for prediction.

B. Addressing Requirement 2: Automated Deployment of Application Components on Heterogeneous Resources

Requirement 2 on automating the deployment on potentially heterogeneous resources is also addressed using Stratum's DSML and its generative capabilities. We captured the infrastructure specification details of the data analytics pipeline in

the metamodel [14]. The user models the deployment scenario by using the intuitive abstractions using the Stratum GUI. Then, the Stratum *model interpreter* verifies the correctness of the abstract deployment model. A code generator within the Stratum DSML generates the Infrastructure-as-Code (IAC) solution by parsing the user-defined model and deploying it on the target machine using a template-based transformation and knowledge base. Finally, the IAC solution is executed by the *code executor* to deploy the desired data analytics architecture on the target machines across the cloud-edge spectrum as shown in Figure 1. The deployment is accomplished via the Stratum MDE approach without writing a single line of code.

1) Metamodel for Data Ingestion Frameworks: The metamodel for data ingestion tools (e.g., RabbitMQ, Kafka) capture the details of services for interacting with the Data Repositories and other microservices using RESTful APIs. The user must select the specific data ingestion tool to deploy it on the target platform. We also verify the correctness of the user model based on the semantics defined in the metamodel. For example, if the user selects the AzureEventHubs as there desired Data Ingestion Tool, then Amazon AWS or OpenStack or bare metal cannot be its target deployment platform.

2) Metamodel for Data Analytics Applications: The live or batch analytics frameworks such as Hadoop, Spark, Flink etc. need to be deployed on the target distributed systems. Moreover, the trained ML model needs to be integrated with these data analytics frameworks. So to deploy the production-ready machine-learning pipeline, we captured the specifications for the ML libraries and frameworks such as Tensorflow, scikit learn along with live and batch analytics frameworks.

3) Additional Metamodels: Stratum also provides metamodels to capture the heterogeneity in resources (e.g., Raspberry Pi, NVIDIA Jetson TX1, bare metal server machines, etc), different operating systems and versions, as well as data stores (e.g., AmazonS3, HDFS, AzureBlobStorage, Ceph).

C. Addressing Requirement 3: Framework for Performance Monitoring and Intelligent Resource Management

Once the trained models are deployed and start executing, runtime resource management is required. Stratum supports autoscaling and load-balancing using the serverless paradigm.

1) Performance Monitoring: It is critical to monitor the system metrics to understand the runtime performance of the infrastructure and the application. Stratum leverages our recent work on FECBench [15] to collect different system metrics, such as GPU-specific metrics such as power consumption, GPU utilization, temperature, and host-specific metrics such as CPU, disk, network, low-level cache utilization, memory bandwidth utilization, using *CollectD* at different granularities from a distributed set of resources and collect these information at a centralized server using *InfluxDB* .

2) Resource Management: Stratum contains a *Resource Manager* to maintain the QoS of the application components by scaling and migrating the application components. The latency of predictive analytics applications is the summation of round trip latency (l_{rt}) of data and the ML model execution time. We profile the ML model execution times on various data and target hardware before actually deploying the model, and consider its 95^{th} percentile execution latency as estimated execution time $(exec_{hw\_id,mlmodel})$.

Migration of ML Prediction Tasks. Before considering edge devices as a potential node for executing predictive analytics, we check if it has sufficient memory to keep the model in memory. If the edge node can host the ML model, we profile the ML model on the edge devices. We also profile 95^{th} percentile network latency (l_{rt}) between edge and cloud node. We consider the migration of the ML model in the edge if $exec_{edge,mlmodel} < exec_{cloud,mlmodel} + l_{rt}$.

Auto-scaling of Application using Serverless Paradigm: Let χ denote the constraint specified by the Service Level Objective (SLO) of the ML model execution latency, and let $exec_{hw\_id,mlmodel,p}$ denote 95^{th} percentile execution latency on p CPU cores. For each server configuration, we can compute the number of requests n_req it can serve for a prediction service while meeting the SLOs using p CPU cores. By monitoring the total number of incoming requests, we calculated the total number of machines $(total\_incomingrequests/n_{req})$ required to handle the highly-parallel workloads is. We calculated how many more machines to provision based on the difference between the current state and desired state. The Stratum *Resource Manager* deploys the ML model on these machines and starts the prediction service automatically to handle dynamic workload proactively [16].

D. Discussion and Current Limitations

Currently Stratum is capable of generating only Python based code, and only Scikit-learn and TensorFlow are integrated. However, other languages such as Java, C++ can easily be plugged into Stratum, and other cloud libraries such as Amazon SageMaker, AzureML can be integrated with Stratum very easily. The design of Stratum uses agile methodologies so that it can be extended with ease. We also incorporated the automatic version control in Stratum framework so that we can recall a specific version of the framework later if required.

V. Evaluation of Stratum

In this section we evaluate the simplicity, rapid deployment, and resource management capabilities of Stratum, along with the accessibility, scalability, and efficiency of the ML model development framework of Stratum.

A. Evaluating the Rapid Model Development Framework

We show how Stratum's MDE capabilities eases ML developement. Figure 2b shows how the ML developer can build their ML pipeline using the visual interface of Stratum. In the left-hand pane (box1), all the building blocks are defined using the metamodel. The ML model developer has to drag and drop the required blocks in the design pane (box 2) and must connect the blocks to define the ML pipeline including preprocessing, ML algorithm selection, hyperparameter tuning, model evaluation, and best model selection criteria. All the attributes of the selected ML algorithms such as max_depth,

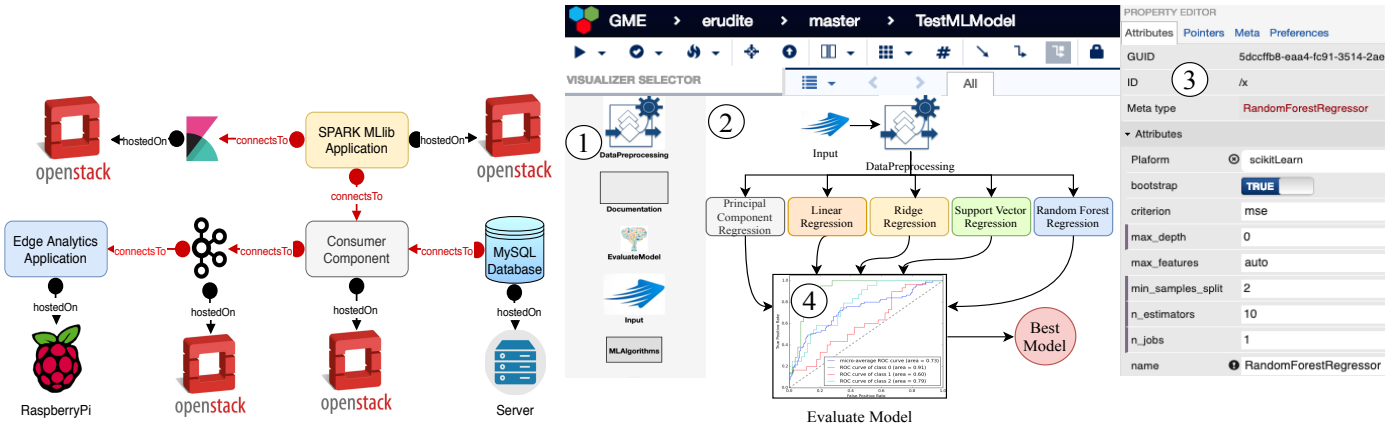

Fig. 2: (a) Example of Big Data Analytics System Deployment Model across Cloud/Edge Spectrum. (b) Usability of the ML model development Framework. Box 1 shows the available selection of blocks available to create ML pipeline as shown in Box 2. Attributes can be set using attribute selection panel in Box 3. Box 4 shows model evaluation.

criteria, etc need to be specified by the user (or can take default values) from the right pane (box 3). The name of the attributes are dependent on ML algorithms, and this aspect is captured by reverse engineering. The ML execution framework needs to be mentioned to bind the workflow with a specific library or framework such as Scikit-learn, Spark MLlib, or Tensorflow.

All the ML algorithms are encapsulated in Docker containers, and different algorithms can be executed in parallel to speed up the training and tuning phases. Similarly, in the input building block, the source data type, and path needs to be mentioned, and also data source type, e.g., csv, text is required. In the data preprocessing block, we only support simple data cleaning methods, such as filtering and normalization. In the evaluation building block, the method of evaluation needs to be specified, and based on that, Stratum selects the best model.

The model transformer can distribute different jobs with different ML algorithms over a cluster of connected machines and aids the developer to select the best model or ensemble of models based on the user's choice of evaluation methods. A sample ROC curve shown in box 4 depicts that the ensemble of two ML methods has the highest accuracy in the training phase on a sample dataset, and it should be selected as the best model. Thus, Stratum helps to build the ML model using MDE techniques, and the ML developer does not need to write any code on supported ML frameworks. The framework can also generate the code in the notebook environment for the expert user, where they can tune the ML model as required.

B. Evaluation of Rapid Application prototyping Framework

As depicted in Figure 2a, using the visual interface of Stratum the application developer can develop the data analytics application. As described in our previous work [14], the business application workflow is designed by dragging and connecting the specific building blocks for application components and infrastructure components. As shown in Figure 1, by parsing the user-defined abstract model tree, the Stratum DSML creates the deployable model (Ansible-specific in our case), and using NodeJS based plugin to execute the deployable model and create the infrastructure of the application as described in Section. IV-B.

Figure 2a illustrates that using the Stratum modeling environment, edge analytics application components can be deployed on Raspberry Pi machine, and data ingestion tool, e.g., Kafka can be deployed on cloud VMs, which is maintained by OpenStack. The data consumer application component can be similarly deployed on OpenStack VM, which will consume the data from Kafka in a batch or stream and store it in MySQL database, which is deployed on top of the bare-metal server. Then, Spark with the MLlib library can be deployed and configured on OpenStack VMs, and a visualization engine like Kibana can be integrated with the workflow. RESTful APIs connect all the application components, so the ML model can easily be pushed into predictive analytics application during management phase. We developed a sample big data use case scenario using Stratum for *sentiment analysis of Twitter data stream* using Figure 2a deployment model.

C. Performance Monitoring on Heterogeneous Hardware

As described in Section IV-C1, to describe the monitoring capabilities of Stratum, we set up the training experiments on NVIDIA GeForce Titan X Pascal GPU machine integrated with Intel(R) Xeon(R) CPU E5-2620 v4. For prediction experiments, we set up a cloud cluster of Intel(R) Xeon(R) CPU E5-2620 v4 machines, Dell OptiPlex 3020 machines, and MinnowBoard with 64-bit Intel Atom devices as edge devices.

Deep Learning Model Training for Image Classification: During the training phase of a deep learning model for image classification using CIFAR10 dataset, we monitor the accuracy and loss of the model along with the GPU performance metric such as GPU utilization, GPU memory utilization per core, the power drawn by GPU cores in watt, and the temperature of the GPU machine in Celsius as shown in Figure 3.

Prediction using Pretrained Image Classification model: Stratum enables us to monitor the performance of the ML containers along with host machines. We collect various metrics, which includes execution time, CPU, memory, network,

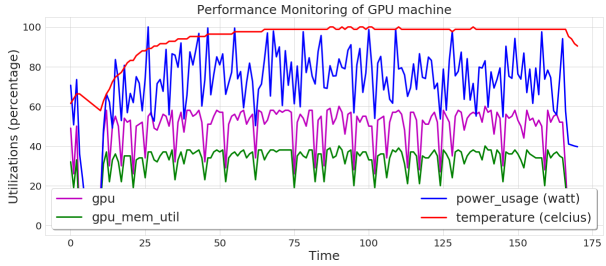

Fig. 3: GPU performance metrics for Deep Learning Training

disk utilization along with L2, L3 cache bandwidth etc. Figure 4 illustrates a glimpse of collected performance metrics during the prediction serving phase. Figure 4(a) shows the execution latency of InceptionResnetV2 and Xception models on different ML containers with variable configurations, which is hosted on different bare-metal machines. Figure 4(b) shows CPU utilization of the ML containers from the host machines,and Figure 4(c) shows memory utilization of ML containers (in MB) from host machines.

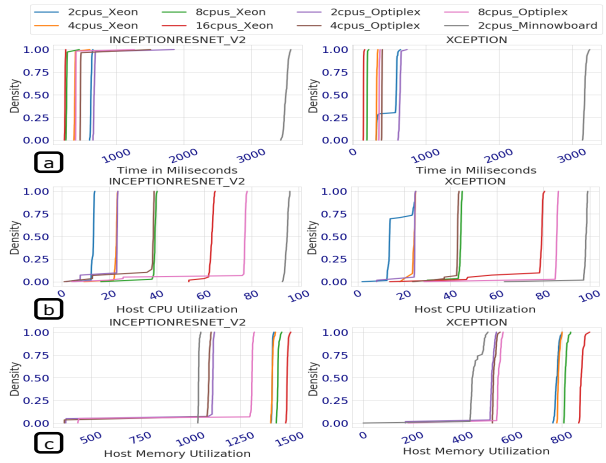

Fig. 4: (a) Latency of InceptionResnetV2 and Xception prediction services, (b) Host CPU utilization, (c) Host Memory utilization

D. Resource Management

As mentioned in Section. IV-C2, we profile the prediction service on the specific hardware before deploying it on the cluster of machines. Based on the number of incoming requests (dynamic workload), we scale up and down the number of ML model containers to guarantee the pre-defined QoS in an event-driven manner using the Docker swarm cluster management tool, as demonstrated in our recent work [16].

VI. Conclusions

As IoT-based analytics becomes increasingly sophisticated, developers are finding themselves lacking expertise in a wide range of skills while also being overwhelmed by the plethora of frameworks, libraries, programming languages, and hardware available to design and deploy analytics applications. To address these highly practical challenges, this paper presents Stratum, which is a novel Big Data-as-a-Service for the lifecycle management of IoT analytics applications. Stratum provides a graphical tool that allows a user to graphically compose the ML development and deployment pipeline using its supported features, and Its DSML realizes the model and generate the code to automate and orchestrate the application lifecycle management across the cloud-edge spectrum.

The Stratum framework capabilities are available for download from github.com/doc-vu/Stratum.

Acknowledgment

This work was supported in part by AFOSR DDDAS FA9550-18-1-0126 and AFRL/Lockheed Martin StreamlinedML program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFOSR or AFRL.

References

[1] E. Blasch, S. Ravela, and A. Aved, *Handbook of dynamic data driven applications systems*. Springer, 2018.

[2] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[3] S. Shekhar, A. D. Chhokra, A. Bhattacharjee, G. Aupy, and A. Gokhale, "Indices: exploiting edge resources for performance-aware cloud-hosted services," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 2017, pp. 75–80.

[4] D. C. Schmidt, "Model-Driven Engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[5] K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Reading, MA: Addison-Wesley, 2000.

[6] A. Bhattacharjee, Y. Barve, S. Khare, S. Bao, A. Gokhale, and T. Damiano, "Stratum: A serverless framework for the lifecycle management of machine learning-based data analytics tasks," in *2019 USENIX Conference on Operational Machine Learning (OpML 19)*. Santa Clara, CA: USENIX Association, May 2019, pp. 59–61. [Online]. Available: https://www.usenix.org/conference/opml19/presentation/bhattacharjee

[7] T. Li, J. Zhong, J. Liu, W. Wu, and C. Zhang, "Ease. ml: Towards multi-tenant resource sharing for machine learning workloads," *Proceedings of the VLDB Endowment*, vol. 11, no. 5, pp. 607–620, 2018.

[8] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc *et al.*, "Tfx: A tensorflow-based production-scale machine learning platform," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1387–1395.

[9] (2017) Meet michelangelo: Uber's machine learning platform. [Online]. Available: https://eng.uber.com/michelangelo/

[10] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe *et al.*, "Accelerating the machine learning lifecycle with mlflow," *Data Engineering*, p. 39, 2018.

[11] Y. Lee, A. Scolari, B.-G. Chun, M. Weimer, and M. Interlandi, "From the edge to the cloud: Model serving in ml .net," *Data Engineering*, p. 46, 2018.

[12] K. Kumar, S. Sinha, and P. Manupriya, "D-pnr: Deep license plate number recognition," in *Proceedings of 2nd International Conference on Computer Vision & Image Processing*. Springer, 2018, pp. 37–46.

[13] M. Maróti, R. Kereskényi, T. Kecskés, P. Völgyesi, and A. Lédeczi, "Online collaborative environment for designing complex computational systems," *Procedia Computer Science*, vol. 29, pp. 2432–2441, 2014.

[14] A. Bhattacharjee, Y. Barve, A. Gokhale, and T. Kuroda, "A model-driven approach to automate the deployment and management of cloud services," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 2018, pp. 109–114.

[15] Y. D. Barve, S. Shekhar, A. Chhokra, S. Khare, A. Bhattacharjee, Z. Kang, H. Sun, and A. Gokhale, "Fecbench: A holistic interference-aware approach for application performance modeling," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*, June 2019, pp. 211–221.

[16] A. Bhattacharjee, A. D. Chhokra, Z. Kang, H. Sun, A. Gokhale, and G. Karsai, "Barista: Efficient and scalable serverless serving system for deep learning prediction services," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*, June 2019, pp. 23–33.