Can Model Driven Help Machine Learning? A Practice of AutoScaling Cloud Services

Hanieh Alipour
Electronic and Computer Engineering
Concordia University
Montreal, Canada
Email: h_alipou@encs.concordia.ca

Yan Liu
Electronic and Computer Engineering
Concordia University
Montreal, Canada
Email: yan.liu@concordia.ca

***Abstract*—Autoscaling is a cloud process that reacts to real-time metrics and adjusts service instance counts accordingly based on predefined scaling policies. However, such a reactive approach still lags behind fluctuating load changes. To address this problem, the learning-and-inference based prediction has been adopted to predict the needs prior to provision and act accordingly. New questions further arise: (1) How to learn from the real-time metrics used for autoscaling for workload prediction? (2) How to integrate the machine learning workflow with the autoscaling processing? (3) How to reduce the deployment effort of devising additional machine learning components? In this paper, we present a model-driven framework that defines first-class entities to represent machine learning algorithm types, inputs, outputs, parameters, and evaluation scores. We capture the relations of machine learning entities; we define new data types and setup rules of validating the model properties. The connection between machine learning and cloud platforms is presented by two level of abstraction models, namely cloud platform independent model and cloud platform specific model. We automate the model-to-model transformation and model-to-deployment transformation by means of DevOps tools. We evaluate our practice of this work by scaling the Netflix benchmark on two cloud platforms, AWS and Azure. The evaluation shows our inference-based autoscaling with model-driven reduces approximately 27% of effort compared to ordinary autoscaling.**

***Index Terms*—Model-Driven, Autoscaling, Machine Learning, Cloud Computing, DevOps**

I. Introduction

One of the essential cloud services is an autoscaling service. The autoscaling service allows for the provisioning of virtualized resources given a specific condition, that can be schedule-based, event-triggered or threshold value based. These condition-triggered autoscaling mechanisms are reactive rather than predictive. This means that the decisions of autoscaling actions are based on previous metrics, rather than a predictive action based on the forecasting of the workload. The machine learning techniques need to be combined with the autoscaling process in order to scale, based on the prediction results. Also, the cloud service providers offer different techniques for configuration and deployment of their services such as autoscaling and machine learning services. Thus, the heterogeneity increase the complexity of implementation. Deployment in the cloud is a complex process that requires a substantial number of tasks to finish successfully. Furthermore, to successfully deploy a service such as autoscaling, proper preparation, and knowledge of the target cloud platform is required. The cloud providers offer different APIs to manage resources; this API heterogeneity enforces an expensive learning requirement for developers. Due to the increasing complexity of the cloud services, the human effort also is increasing. The growing complexity of cloud services is the motivation behind work on automatic configuration and development. So, there is a need to minimize the human effort and automate the deployment.

Addressing these challenges requires a solution that focuses on the core functions of the autoscaling process and key elements of machine learning. A primary goal of using the model-driven approach is to propose a solution that shields developers from the complexities associated with details of the underlying target cloud platform. Each cloud service may support different type of APIs or programming languages. The goal is to hide the complexities of using different technologies from developers who are responsible for managing the cloud environment. In other words, the human effort is based on the effort required to tackle the considerable complexities arising from the use of different cloud technologies and services. Moreover, minimizing human effort is extremely important in practice since deploying and configuring of cloud services can be expensive.

Thus, the model-driven development principle is used to describe the autoscaling process at the service level, and the mapping to the cloud platform-specific configuration is automatically generated and deployable. In this approach, models drive the key elements of machine learning, and these models are specified at two levels of abstraction, namely a Cloud Platform Independent Model (CPIM) and a Cloud Platform Specific Model (CPSM). Furthermore, automated tools are applied to model transformations. The model-driven approach reduces the gap between problem domains and service implementation with technologies that support the efficient transformation of the abstract model to service implementations. Therefore, we develop a model-driven approach to address the challenges outlined above. In this context, we propose an abstract model, based on the model-driven standards, to model predictive autoscaling service deployment and to enable prediction of future demand. This work allows the generation of automatic scripts from models, which are then deployed on the target cloud platform.

Detailed below we have described four research questions that our work covers:

- RQ1) How to model a machine learning service? We propose a solution to calculate the multiple metrics mixture effect and model the key elements of machine learning (**Section III**).
- RQ2) How to integrate the machine learning with the autoscaling process? We proposed model-driven models for machine learning service with the autoscaling process (**Section IV**).
- RQ3) How to make model deployable and executable into the autoscaling process in run-time? We integrate the model-driven solution with a DevOps approach (**Section IV-B**, **Section IV-C**).
- RQ4) How to evaluate the effectiveness of the proposed approach? We calculate the effort in term of the impact of changes in multi-clouds (**Section V**).

With this approach, our study has identified three contributions:

- We propose a high-level abstraction for modeling the particular features of the machine learning such as machine learning algorithm types, inputs, outputs, parameters, and evaluation scores. This abstraction helps to capture functionality and computation of machine learning service while hiding cloud providers details. Furthermore, we propose the new data type metric to meet the machine learning requirements.
- We integrate a machine learning model with the autoscaling process and introduce the predictive autoscaling model in two abstraction levels. We design the CPIM and the CPSM which they are focused on the autoscaling process and machine learning model. The result maps the predictive autoscaling entities that are general across multiple clouds at the abstract level. A CPSM model is further customized for specific predictive autoscaling features of a cloud service provider.
- We develop a method to integrate the model-driven solution with a DevOps approach. Thus, the result of this integration facilitates the deployment actions and make the proposed models deployable and executable on the target cloud platform. We calculate the deployment effort for our proposed solution and compare it with the effort of regular autoscaling and machine learning service on two different cloud platforms to demonstrate the improvement.

We have designed a specific test scenario for estimating the effort of the predictive autoscaling model on two cloud platforms. Based on the evaluation result, we observe the reduced effort using our model-driven method.

II. Related Work

Sun et al. [2] proposed a ROAR modeling framework to automate and optimize the testing and derivation of cloud resource allocation for Web applications to meet the QoS goal. The end-to-end test orchestration of resource allocation, load generation, resource utilization metric collection, and QoS metric tracking (delay, throughout) performs automatically. In addition, they support the deployment of an application to multiple cloud providers. They mainly focus on optimizing load testing for resource allocation and enhancing the existing cloud deployment. They do not introduce how to scale out or scale in based on unexpected load. There is still a lack of generalization of autoscaling mechanisms in different cloud environments.

Song et al. [3] used the Exponentially Weighted Moving Average (EWMA) model for predicting the demand for the number of virtual machines. They proposed an online bin packing approach that uses virtualization technology to allocate cloud resources dynamically based on application demands. Their proposed approach supports green computing by optimizing the number of servers that are used. Bunch et al. [4] designed provisioning systems that predict the future service demand to decide the number of resources that are needed for provision. They developed a pluggable and cost-aware autoscaling system that forecasts the future demand by analyzing metrics such as the request volume.

Gandhi et al. [5] presented Dependable Compute Cloud (DC2) as a new cloud service. In their model-driven autoscaling approach, they combined a Kalman filtering technique and queuing theoretical model in DC2 to choose the right scaling action. However, they did not consider multiple clouds and the vendor lock-in issue in their approach.

MODAClouds [6] followed a model-driven approach to design and execute applications on multiple clouds to support interoperability and prevent vendor lock-in. They provided the automatic deployment of applications in various clouds with guaranteed QoS. The primary goal of MODAClouds was to support migration applications from cloud to cloud as needed. MODAClouds considers three levels of abstraction: the Cloud Enabled Computation Independent Model to describe an application and its data, the Cloud Provider Independent Model to describe cloud concerns related to the application in a cloud-agnostic way, and the Cloud Provider Specific Model to represent the cloud concerns that are associated to deploy and provision the application. However, their work does not focus on autoscaling mechanisms in cloud computing.

Sharma et al. [7] studied the MDA approach to developing software systems. They highlighted the benefit of incorporating the model-driven architecture in the development of cloud SaaS to minimize time, cost and effort. Eldein et al. [8] studied how to use model-driven architecture, discussed open issues and explained future research problems. In fact, this paper analyzed the research challenges that have been emerging in cloud computing and the model-driven development.

Ferry et al. [9] proposed a model-based framework named the Cloud Modelling Framework (CloudMF). They employed MDE to face the complexity of developing complex multiple cloud systems. They also introduced CloudML, which relies on model-driven techniques. It is domain-specific modeling language, and it facilitates the specification at the design-time of provisioning and deployment.

The DevOps approach has been investigated for its combination with model-driven development to improve the quality of service for a complex system. Wettinger et al. [10] presented a concept that integrates model-driven and configuration management using Chef. The Chef is an agent-based framework, and it requires master-agent communication. In our work, we chose Ansible to demonstrate our work. It is simplified in terms of communication via standard SSH commands. Bruneo et al. [11] introduced the CloudWave that employs DevOps to create an execution analytics cloud infrastructure to obtain high QoS levels. Their goal was to improve both the development of SaaS solutions and the management of their operation and execution.

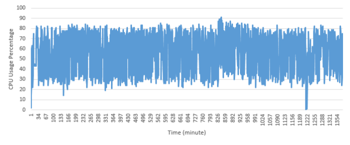
III. Problem Statement

In our previous work [12], we presented a model-driven method to connect a cloud platform independent model of services with cloud specific operations. Through the automated transformation from model to the configuration, we used cloud management tools to deliver autoscaling deployment across clouds. The experiment demonstrates our proposed method resolves the vendor lock issues through model-to-configuration-to-deployment automation. On the other hand, recently we developed [13] a novel machine learning-based autoscaling process that covers the technique of learning multiple metrics for cloud autoscaling decision. This technique is used for continuous model training and workload forecasting. Furthermore, the result of workload forecasting triggers the autoscaling process automatically. Based on our previous works, there is a need to model key elements of machine learning service and integrate the machine learning model and the autoscaling process.

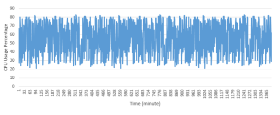
A. Multi-Variate Analysis

When we have multiple metrics, each metric may have different effects on the behavior of the system. In order to calculate the effects, we need to compute the weight of each metric and find multi-metrics mixture effect. Multiple Attribute (metric) Decision Making (MADM) refers to making decisions in the presence of multiple metrics. In line with the MADM concept, our concern is to build the machine learning model involving multiple metrics and finding appropriate weight for each metric. Shannons entropy method is one of the various methods for finding weights discussed in the literature [14], [15]. The Shannon entropy weight, which expresses the relative intensities of metric importance. Our work presents how we solve the problem via analysis of metrics weights by mean of entropy. In this research, we monitored one sample per minute for each metric and thus 20160 samples for two weeks. We collected a total of 20160 samples of multiple metrics for training. Figure 1 demonstrates the example of the workload of multiple metrics and the final result of aggregated metrics.

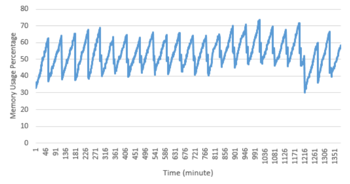
Moreover, our proposed solution calculates the multiple metrics mixture effect based on the weight average of metrics.



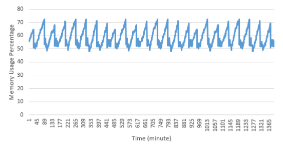
(a) CPU usage on Azure



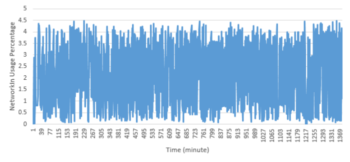
(b) CPU usage on AWS



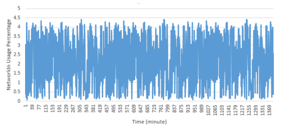
(c) Memory usage on Azure



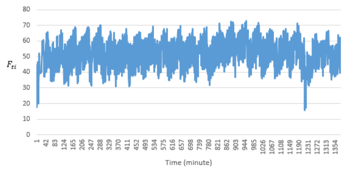
(d) Memory usage on AWS



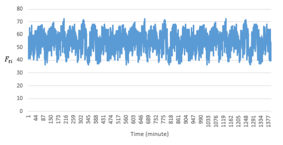
(e) NetworkIn/Out usage on Azure



(f) NetworkIn/Out usage on AWS



(g) F_{t_i} on Azure



(h) F_{ti} on AWS

Fig. 1: Real-time system metrics for autoscaling. Each metric consists of 1377 records from 20160 samples (two weeks)

We suppose there are m candidate records in T time period with M number of metrics. The four metrics are *CPU*, *Memory*, *NetworkIn* and *NetworkOut*. Following steps demonstrate our solution for calculating the weight of each metric.

Step 1: Normalize the records of each metrics:

$$N_{t_i,M_l} = R_{t_i,M_l} / \sum_{i=0}^{m} R_{t_i,M_l} \tag{1}$$

$$t_i \in [0, T]$$

$$M_l \in [CPU, Memory, NetworkIn, NetworkOut]$$

Where T is 20160 samples for two weeks and M is the size of M_l , so $M = 4$.

Step 2: Calculate Entropy for each metric as :

$$E_{M_l} = (-K) \sum_{t=0}^{T} (N_{t_i,M_l} \ln(N_{t_i,M_l})) \tag{2}$$

K is the entropy constant as

$$K = 1/ln(m)$$

Step 3: Compute the degree (weight) of importance of metrics as:

$$W_{M_l} = 1 - E_{M_l} / \sum_{M_l=0}^{M} (1 - E_{M_l}) \tag{3}$$

When we have the weight of each metric, we use the weight average metric method to calculate the multiple metrics mixture effect F_{ti}. Therefore, metric with a higher weight contributes more to the weighted mean than metrics with a lower weight. Hence, F_{ti} becomes the inputs of the machine learning algorithms for training the history workload and predict the future workload.

$$F_{t_i} = \sum_{M_l=1}^{M} (W_{M_l} * N_{t_i,M_l}) / \sum_{M_l=1}^{M} (M_l) \tag{4}$$

This weighted metrics aggregation method is extensible to a variety of metrics. This method allows us to combine application level metrics and system-level metrics as machine learning inputs.

B. The Needs of Modeling Machine Learning

Key Machine Learning Elements: The machine learning service includes three main phases: the preparing and analyzing data, the training and evaluating models, and prediction. Each cloud provider offers different techniques and API for those phases, so there is a need to have an abstract model which hide the complexities associated with details of the underlying cloud platform. In Table I, we list three different machine learning algorithms and demonstrate key elements of the machine learning algorithms. There are the common elements across algorithms as version, input, output, tuning parameters, and score. We consider the input as the weighted metric aggregation value of F_t in time series. The output is values of the parameters trained. The accuracy of the algorithms is measured by performance scores.

Tuning parameters differs from the model parameters in that they are not learned by the model automatically through training. Instead, these parameters such as learning rate are related to how fast the machine learning model converges for a solution. Therefore, tuning parameters should be explicitly modeled. Each time we change a tuning parameter, we have a new instance of the machine learning model. To distinguish each setting of turning parameters tried, we introduce version as an essential attribute for modeling in addition to the intrinsic elements of machine learning as inputs, outputs, and scores.

New Data Type: The numeric values of inputs, outputs, tuning parameters, and scores are of different types and scales. The values of inputs are time series, and ordering is essential; while the value of performance scores are singular real numbers. The outputs are trained values of model parameters that are also real numbers. The output also contains inference results as the probability that are real numbers between 0 to 1. To customize these variations of data types, one possible solution is using the default data types with constraints attached. However, this way lacks of explicity in the first order of modeling, as the characters of the data type depending on the

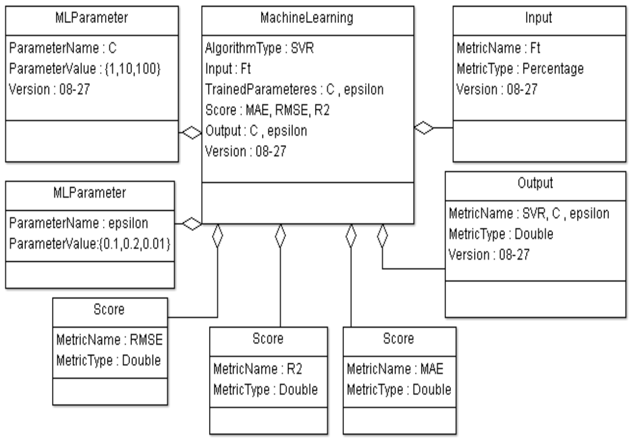


Fig. 2: Example of Cloud Platform Specific Model for Machine Learning Object

TABLE I: Key Elements of Machine Learning Algorithms for Workload Forecasting

Algorithm Type	Version	Input	Tuning Parameters	Output	Score
Support Vector Regression	V_{ts}	F_t	-C (Penalty Factor) -$Epsilone$ (Margin of Tolerance)	-Model ID -Parameters	-MAE -$RMSE$ R^2
Gradient Boosting Regression	V_{ts}	F_t	-$N\_estimators$ (Limit number of trees) -$Learning\_rate$ (How quickly the error is corrected)	-Model ID -Parameters	-MAE -$RMSE$ -R^2
Linear Regression	V_{ts}	F_t	– –	-Model ID -Parameters	-MAE -$RMSE$ -R^2

constraints defined behind. Instead, we introduce a data type, called *Metric* for representing metrics of which each attribute is of a specific type. The attribute *Percentage* is of double type in the range of 0 to 1. The *Byte* attribute has the type of positive long values. The attribute *Double* is used for values of parameters and scores. Finally, the attribute *BytePerSec* is used to capture metrics of rates, such as *NetworkIn* and *NetworkOut* system level network communication metrics.

Rules: The *MachineLearning* class is a composition class of composing classes such as *Input*, *Output*, *MLParameter*, and *Score*. We have discussed the fact that any variation of tuning parameters produces a new instance of machine learning. Therefore, tracking the instance linkages and ensuring consistency becomes a mandatory requirement. Such a requirement is defined as rules for checking instance consistency by the attribute of *Version*.

Assume we have three different sets. $S_1 = \{x_1, x_2, x_3, -, x_n\}$ contains all the modeling instances and $S_2 = \{y_1, y_2, y_3, -, y_n\}$ is a set of composition instances. In addition, we have $S_3 = \{z_1, z_2, z_3, -, z_n\}$ which represents the all attributes. There is a rule that each instance which has composition instance have to have the same attribute (as in Rule 1).

$Rule1:$
$\exists x \in S_1, \forall y \in S_2, \forall z \in S_3, Instance(x) \wedge Composition(y),$
$Instance(z) \equiv Composition(z)$

For example, it is essential to keep consistency between the versions of machine learning algorithm elements. The version of machine learning instance should be equivalent of the version of the input, the output, and the tuning parameters.

$Example:$

$\exists x \in S_1, \forall y \in S_2, \forall z \in S_3, Instance(x) \wedge Composition(y),$
$MachineLearning(V_{ts}) \equiv MLParameter(V_{ts})$

C. Integration of Model-Driven and DevOps Approach

Models should be transformed into deployment entities to help reduce deployment effort. Otherwise, models remained at the design phase and isolated from the rest of the lifecycle of the autoscaling process. DevOps is an emerging paradigm to actively foster the collaboration between system developers and operations in order to enable efficient end-to-end automation. DevOps is typically combined with cloud computing, which allows rapid, on-demand provisioning of underlying resources such as virtual servers, storage, or database instances using APIs in a self-service manner. The goal is to bring together the strengths of DevOps and model-driven approach in order to minimize the effort. So, models are transformed into scripts of DevOps tools. Hence, the deployment, configuration, and trigger of cloud services are carried out by DevOps tools. We consider Ansible, a configuration management and provisioning tool (DevOps) that supports public and private cloud platforms including AWS, Google Cloud Engine, Microsoft Azure, OpenStack, and Rackspace.

IV. Model-Driven Deployment Method

The proposed model-driven method consists of three main phases: Design, Deployment, and Run-time phases (Figure 3).

A. Design Phase

The *Design phase* includes the model-to-model transformation. This phase contains two levels of abstraction, the Cloud Platform Independent Model (CPIM) and the Cloud Platform Specific Model (CPSM). The CPIM in a *ecore* model follows the model-driven principles to create an abstract representation of the knowledge and activities in the context of machine learning service and an autoscaling process. We integrate the machine learning model with the proposed autoscaling model in [12]. In this paper, we build a new meta-model shown in Figure 4. The CPIM represents the main predictive autoscaling components. The CPIM model hides cloud platforms details and remains at the service level abstraction.

The CPIM is transferred to the model objects which includes class objects and associations that is specified for a target cloud platform. A CPSM model is created as a model instance of the CPIM using the EMF generator function. The CPSM objects require details to be customized for a specific cloud platform. The next step is adding in model objects with their attribute values configured. Upon the creation of a model object, the rule defined at the class level is checked and validated. The *Epsilon (EVL)* validates the CPSM in order to ensure the model generated is correct and complete.

1) Model Validation: We define the set of rules that need to be satisfied in a CPSM. Assume we have three different sets. $S_4 = \{x_1, x_2, x_3, -, x_n\}$ contains all the modeling objects and $S_5 = \{y_1, y_2, y_3, -, y_n\}$ is a set of mandatory attributes. In addition, we have $S_6 = \{z_1, z_2, z_3, -, z_n\}$ which represents the optional attributes. For example, for a machine learning object, we need to specify the name of algorithms and tuning parameters. Otherwise, this object cannot be mapped to a specific deployable service. The parent components are incomplete without the child component. The rule is defined for composition class such as the *MachineLearning* class.

$Rule2:$

$\exists x \in S_4, Parent(x) \wedge Child(x), Parent(x) \leftrightarrow Child(x)$

The third rule is defined for attributes. For mandatory attributes, their values cannot be empty. For optional attributes, missing values are reminded by warnings and default values are provided.

$Rule3:$

$\forall x \in S_4, \forall y \in S_5, Component(x) \rightarrow Attribute(y)$

$Rule4:$

$\forall x \in S_4, \exists z \in S_6, Component(x) \rightarrow Attribute(z)$

2) Model Transformation: A CPSM is the further input of the *Deployment phase*. By the help of *Epsilon (EGL)*, the CPSM is transferred to a set of deployable scripts. The model to deployment transformation is transforming the models of a predictive autoscaling service to scripts. We automate this transformation using Epsilon [16] on top of EMF. Epsilon includes Epsilon Object Language (EOL) and Epsilon Generation Language (EGL) for parsing EMF models, UML models, and XML files to generate text with EGL templates.

- The *EGL template* contains a static text and a dynamic text retrieved from CPIM and CPSM models defined using EMF.
- The *EGL rule* defines the transformation templates and targets.
- The *EGX program* is the driver to execute EGL rules within Eclipse to launch deployment tools to run on the scripts and/or configuration files created using EGL.

B. Deployment Phase

For AWS, we use the AWS *CloudFormation* service to deploy machine learning elements. The *CloudFormation* service includes a template that contains all the information extracted from the CPSM model. When the template is submitted by the Ansible script, the *CloudFormation* service launches the necessary resources such as a *Machine Learning EC2 Instance*, a *Lambda Function*, and a *DynamoDB Storage*.

The following entities are deployed by the Ansible script and the *CloudFormation* service on the AWS:

- Lambda Function: The Lambda function employs a serverless architecture and the code runs without considering managing any servers or a backend service. The Lambda function calls CloudWatch to collect the workload metrics. Also, the Lambda function calls the machine

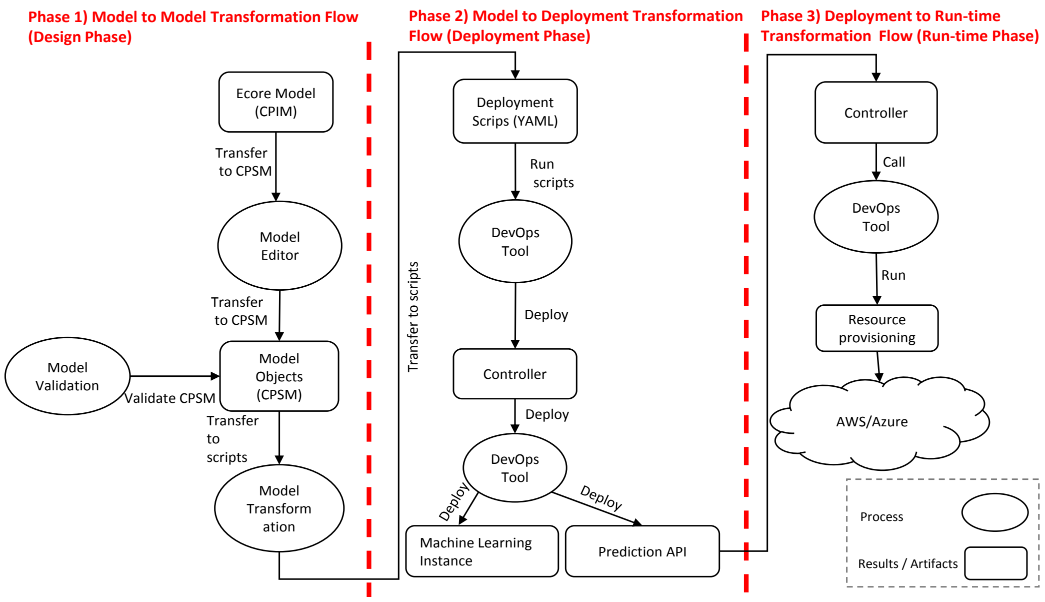


Fig. 3: Model Transformation in the Lifecycle of Autoscaling

learning API to train the models and the predictor for the prediction.

- Machine Learning EC2 Instance: It hosts the training and the prediction services. It is a fully-managed cloud instance for a predictive analytic solution. The workload history is pulled from S3 and utilized by a Machine Learning EC2 instance to train the models.
- Amazon S3 Storage: It stores the metrics collected by CloudWatch.
- DynamoDB Storage: It stores the training results including the performance scores, values of trained parameters, and configuration of the tunning parameters. These values are stored in a NoSQL storage with key/value pairs.

For Azure, we use the *PowerShell* script to deploy the CPSM model. The Ansible script runs the *PowerShell* script, and then the *Azure Function App* is deployed with the necessary codes. The following entities are deployed by the Ansible script:

- Azure Function App: It periodically invokes the Azure Monitoring to collect the workload metrics. Also, the Azure Function App calls the machine learning algorithms to train models. Furthermore, it is the entry to launch the prediction service using a trained model.
- Azure Data Science Virtual Machine: As customized VM image on the Microsoft Azure cloud explicitly built for data science. Furthermore, we implement a REST API to access the inference model as a service for the real-time prediction phase. This service is called by the Azure Function App to perform training and prediction.
- Azure Blob Storage: It contains Blob storage, File storage, and Queue storage. For deployment, we use the Azure Blob Storage. The Azure Blob Storage saves the model training results and the collected metrics as inputs for training.
- Resource Group: It groups Azure instances (resources) for scaling and managing the instances, based on the minimum and maximum number of running instances allowed at any time.

C. Run-time Phase

The run-time phase triggers the execution of deployed machine learning components upon the collection of the system level metrics. The core component to bridge the machine learning components and components of the cloud autoscaling process is the *Controller* component. Figure 5 shows a cloud platform specific model generated for the *Controller* instance on the AWS cloud. The *Controller* is a schedule-based service, and it is trigger by a time interval to call APIs of machine learning services. For example, when the training time is triggered, the *Controller* calls the *MachineLearning* to start model training stage for constructing, testing and validating the models. Each machine learning instance contains one machine learning algorithm. The API is defined on the instance level. The same machine learning algorithm with different tuning parameter values produces multiple instances. Each instance has its unique REST API.

Our proposed solution is based on the aggregation of multiple metrics. These metrics are not used as direct inputs to the autoscaling engine on a cloud platform. Instead, The Ansible playbooks are bundles of actions for the same task. The orchestration of autoscaling components to act on the aggregated metrics are defined in playbooks. After the machine learning prediction, the *Controller* sends the predicted workload to an Ansible playbook that triggers the *PolicyScaling* to generate a scaling command. Finally, the *PolicyScaling* communicates with another Ansible playbook that is called a *ScalingEngine* to take an action of provisioning resources. In a nutshell, at the

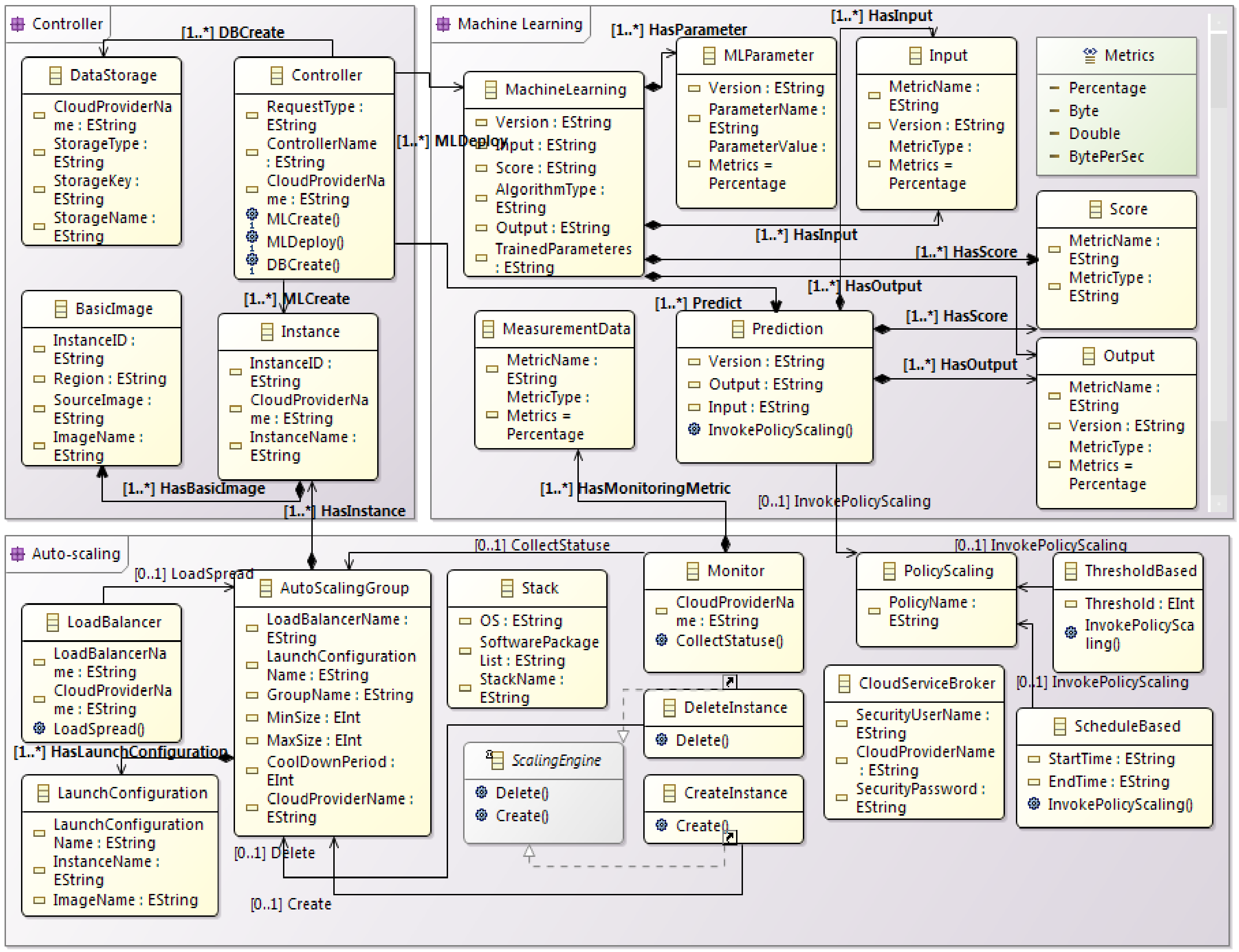


Fig. 4: Cloud Platform Independent Model of Integrated Machine Learning with Autoscaling Service

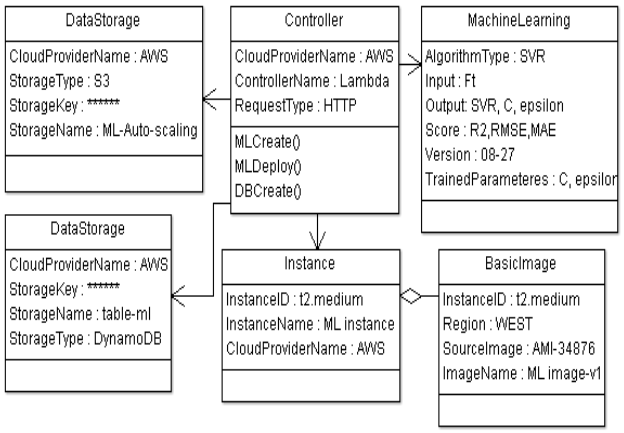


Fig. 5: Example of Cloud Platform Specific Model for Controller

runtime, the orchestration has performed the communication between the *Controller* and Ansible playbooks.

V. The Evaluation

The evaluation of our model-driven framework focuses on the effort in term of the impact of changes required between calling autoscaling service with and without the support of our model-driven framework. We devise a scenario that autoscales data nodes of a benchmark application on two cloud platforms, AWS and Azure. The goal of these experiments is to evaluate the advantage of using model-driven for predictive autoscaling service in cloud computing and to show how we reduce the human effort. We use the Netflix Data Benchmark (NDBench) [17] to evaluate our scenarios. NDBench is designed to examine the performance impact of the backend data systems. NDBench is pluggable for a wide range of cloud providers. We have a separated instance for NDBench that connects to an Apache Cassandra cluster of data storage as the backend system.

We evaluate the effort of our model by the scenario of reconfiguring predictive autoscaling service when the backend cluster of NDBench is migrated from one cloud platform to another. We measure the effort in term of the impact of changes on both clouds. As proposed in the work of Infrastructure as Code (IaC) [20], the cloud deployment process includes five main components, namely installation, configuration, database migration, code modification, and network connection. Since the data node of the NDBench consists of the full stack of packages, the predictive autoscaling service deployment involves building an image of software packages and launching the image on provisioned instances. There are no application code changes required. In addition, for the machine learning service, we employ the Scikit-learn library [18]. It is a free software machine learning library for the Python. The codes are applicable to both cloud environments. There are no code changes required. Our evaluation involves installation, configuration, and model template modification.

The evaluation is structured into three phases, corresponding to the Cloud Migration Point (CMP) [7] approach. During the first phase, the deployment tasks are analyzed to identify and classify the tasks into three categories, namely the *Storage and Database*, the *Template Changes*, and the *Installation and Configuration*. In the second phase, each task is assigned a complexity level, which is determined by the functionality of the cloud services and the interaction of the cloud services with each other. In the last phase, the CMP value is computed as a weighted sum. Based on [7], each function (task) is weighted based on its type and the level of its complexity, in agreement with standard values as specified in the Counting Practices Manual [19].

The evaluation starts with identifying the type of deployment tasks.

- *Storage and Database*- Tasks involved account authentication and establishing connections.
- *Template Changes*- Deployment tasks consist of creating template and model objects. Also, tasks include specification of attributes.
- *Installation and Configuration*- Setup software installation scripts and configure the values of environment variables.

The behavior of each task is taken into account to evaluate its complexity level in terms of the number of methods changed, the number of services edited and the number of attributes modified. The value of CMP is defined as a weighted sum of its three categories CMP_i with $i \in \{$*Installation and Configuration*, *Template Changes*, *Storage and Database*$\}$.

$$CMP = \sum_{i=0}^{2} CMP_i * w_i \tag{5}$$

Where CMP_i is the value of CMP of type i, and w_i is the weighted value for CMP type i.

TABLE II: Complexity Evaluation for Each Installation and Configuration Task

Configuration	Installation		
	Run/No-Installation	Package/Library	Installation
< 2	Low	Low	Average
$2 - 5$	Low	Average	High
$>= 6$	Average	High	High

TABLE III: Weight Evaluation for Each Installation and Configuration Task

CMP	Type	Complexity Level		
		Low	Average	High
CMP_{Ins}	Application	1	2	7
	Infrastructure	1	3	9

A. CMP_{Ins}

The *Installation and Configuration* category (CMP_{Ins}) reviews tasks such as installation of software, server, third-party library and configuration environment variable. All tasks are classified into two types:

- Infrastructure level: Installation of infrastructure level software and servers. For example, setting up an Azure or AWS instance or image, installing OS, and installing the database such as S3 and Blob.
- Application level: Configure application level environment and libraries.

We estimate the complexity level (Low, Average, or High) of each task based on the number of configuration steps and installation type as shown in Table II. Each task is allocated with a weighted value as shown in Table III based on its type and complexity level.

B. CMP_{db}

First, all cloud storage related tasks are grouped into two types, database changes, and API changes. The complexity of each task is determined based on differences between cloud storages and steps for configuring APIs as shown in Table IV. Then, each task is allocated with a weighted value as shown in Table V based on its type and complexity level.

C. CMP_{code}

The *Template Changes* category CMP_{code} assesses all tasks related to create or modify a new model object and a template.

TABLE IV: Complexity Evaluation for Each Storage and Database Task

Storage and Database	Complexity Level
Database Changes	High
API Changes	Average

TABLE V: Weight Evaluation for Each Storage and Database Task

CMP	Type	Complexity Level		
		Low	Average	High
CMP_{db}	Database Changes	1	4	7
	API Changes	1	3	6

TABLE VI: Complexity Evaluation for Each Template Changes Task

Create and Instantiation	Add or Remove Attributes		
	$0-3$	$4-7$	$7-10$
$0-4$	Low	Low	Average
$5-8$	Low	Average	High
$>= 9$	Average	High	High

(a) For Change or Edit Service $(0-2)$

Create and Instantiation	Add or Remove Attributes		
	$0-4$	$5-8$	$>= 9$
$0-3$	Low	Low	Average
$4-7$	Low	Average	High
$>= 8$	Average	High	High

(b) For Change or Edit Service $(2-4)$

Create and Instantiation	Add or Remove Attributes		
	$0-3$	$4-7$	$>= 8$
$0-2$	Low	Low	Average
$3-6$	Low	Average	High
$>= 7$	Average	High	High

(c) For Change or Edit Service $(>= 5)$

TABLE VII: Weight Evaluation for Each Template Changes Task

CMP	Type	Complexity Level		
		Low	Average	High
CMP_{code}	Change or Edit Service	1	2	7
	Create and Instantiation	1	4	9
	Add or Remove Attributes	1	3	6

Three different types are defined to capture aspects of code and template changes:

- Create and Instantiation: Tasks that accommodate creation new model object or template.
- Add or Remove Attributes: Each cloud service requires a set of input (attribute value) and this type cover tasks related to changes in attributes.
- Change or Edit Service: Different cloud services communicate with each other in order to deliver service or functionality.

Based on three types, there are three dimensions to evaluate complexity level as shown in Table VI. Then, each task is allocated with a weighted value as shown in Table VII based on its type and complexity level.

D. The Result

We calculate CMP for the manual deployment procedure for machine learning, manual deployment process of the threshold based autoscaling service, and our model-driven method to deploy the predictive autoscaling service. Figure 6 demonstrates the comparison between the two mentioned scenario for NDBench. We observe the reduced effort is approximately 25.3% for AWS and 26.6% for Azure. All details for the effort calculation are uploaded in http://bit.ly/ICSE-Montreal.

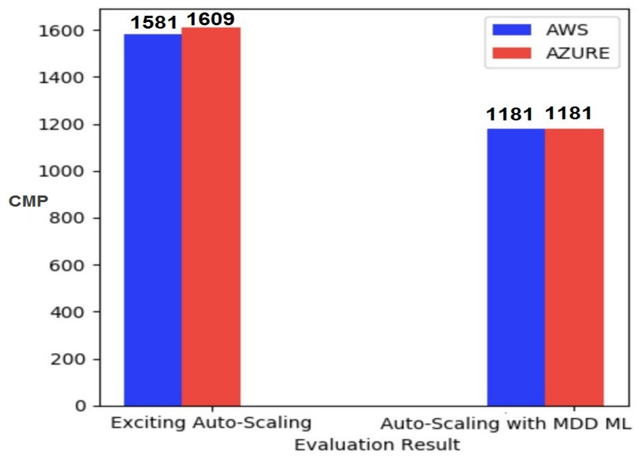


Fig. 6: The CMP result for the Machine Learning, the Threshold based Autoscaling service, and Proposed model-driven Method Deployment Procedure

VI. Conclusion

In this work, we present a model-driven framework to automate the operations of predictive autoscaling service design, deployment and launching on multiple clouds. Our approach contains two parts of models: one for modeling machine learning models independent of a cloud platform; and the other for model cloud-specific autoscaling process. To connect the design level models with the deployment of a cloud platform, we propose the transformation from models to deployment scripts and launch cloud management tools within a single integrated modeling environment. A practical case of this framework is evaluated on both AWS and Azure clouds. Our contribution is to hide the technical details of developing cloud provider specific autoscaling operations with machine learning techniques. Thus, this provides a potential solution to prevent vendor-lock on autoscaling operations.

References

[1] EMF, "https://eclipse.org/modeling/emf/"
[2] Yu Suna, Jules Whitea, Sean Eadeb and Douglas Schmidta, *ROAR: A QoS-Oriented Modeling Framework for Automated Cloud Resource Allocation and Optimization*,Journal of Systems and Software, 2015.
[3] Weijia Song, Zhen Xiao, Qi Chen and Haipeng Luo, *Adaptive resource provisioning for the cloud using online bin packing*,Comput IEEE Trans ,2014.
[4] Chris Bunch,Vaibhav Arora, Navraj Chohan, Chandra Krintz,Shashank Hegde and Ankit Srivastava, *A pluggable autoscaling service for open cloud PaaS systems*, in Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing, 2012.
[5] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang, *Adaptive, Model-driven Autoscaling for Cloud Applications*,in Proceedings of the 11th International Conference on Autonomic Computing, Philadelphia, PA, USA, 2014.
[6] Ardagna, Danilo and et al., *MODACLOUDS: A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds*, MiSE '12 Proceedings of the 4th International Workshop on Modeling in Software Engineering,2012.
[7] Ritu Sharma, Manu Sood, and Chung-Horng Lung, *Enhancing Cloud SaaS Development With Model Driven Architecture*, International Journal of Cloud Comput. Serv. Archit. IJCCSA, 2011.

[8] Amar Ibrahim E.Sharaf Eldein, and Hany H. Ammar, *Model-Driven Architecture for Cloud Applications Development, A survey*, International Journal of Computer Applications Technology and Research, 2015.

[9] Nicolas Ferry, Alessandro Rossini, Franck Chauvel, Brice Morin and Arnor Solberg, *Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems*, IEEE Sixth International Conference on in Cloud Computing, 2013

[10] Johannes Wettinger and et al., *Integrating Configuration Management with Model-Driven Cloud Management Based on TOSCA*, In Proceedings of the 3rd International Conference on Cloud Computing and Services Science(CLOSER), 2013.

[11] Dario Bruneo and et al., *CloudWave: Where adaptive cloud management meets DevOps*, IEEE Symposium on Computers and Communications (ISCC), 2014.

[12] Hanieh Alipour and Yan Liu, *Model Driven Deployment of Auto-Scaling Services on Multiple Clouds*, 2018 IEEE International Conference on Software Architecture Companion (ICSA-C), 2018.

[13] Hanieh Alipour and Yan Liu, *Microservice Orchestration to Inference-based Cloud Workload Auto-scaling*, 2019 IEEE Transactions on Services Computing Special Issue (Submitted).

[14] Yeh, Chung-Hsing,*A problem-based selection of multi-attribute decision-making methods*,International Transactions in Operational Research, volume 9, pages 169–181, 2002.

[15] Karami, Amin, *Utilization and comparison of multi attribute decision making techniques to rank bayesian network options*, 2011.

[16] Epsilon, "http://www.eclipse.org/epsilon/"

[17] Netflix, "https://medium.com/netflix-techblog"

[18] Scikit-Learn, "http://scikit-learn.org/stable/"

[19] Counting Practices Manual, "http://www.ifpug.org/"

[20] Yujuan Jiang and Bram Adams, *Co-evolution of Infrastructure and Source Code: An Empirical Study*, 12th Working Conference on Mining Software Repositories, MSR 15, 2015.

[21] Van Tran, Kevin Lee,Alan Fekete, Anna Liu, and Jacky Keung, *Size Estimation of Cloud Migration Projects with Cloud Migration Point (CMP)*, 2011 International Symposium on Empirical Software Engineering and Measurement. IEEE, 2011.