Review Article
Volume 10:7, 2021

International Journal of
Sensor Networks and Data Communications

ISSN: 2090-4886 Open Access

Function-As-A-Service Acceleration for IoT Applications on Hybrid Cloud

Michael Zhang*, Chandra Krintz and Rich Wolski

Department of Computer Science, University of California, Santa Barbara, United States

Abstract

Serverless computing is an emerging event-driven programming model that accelerates the development and deployment of scalable web services on cloud computing systems. Though widely integrated with the public cloud, serverless computing use is nascent for edge-based, IoT deployments.

In this work, we present STOIC (Serverless Tele Operable Hybrid Cloud), an IoT application deployment and offloading system that extends the serverless model in three ways. First, STOIC adopts a dynamic feedback control mechanism to precisely predict latency and dispatch workloads uniformly across edge and cloud systems using a distributed serverless framework. Second, STOIC leverages hardware acceleration (e.g. GPU resources) for serverless function execution when available from the underlying cloud system. Third, STOIC can be configured in multiple ways to overcome deployment variability associated with public cloud use. We overview the design and implementation of STOIC and empirically evaluate it using real-world machine learning applications and multi-tier IoT deployments (edge and cloud). Specifically, we show that STOIC can be used for training image processing workloads (for object recognition) – once thought too resource-intensive for edge deployments. We find that STOIC reduces overall execution time (response latency) and achieves placement accuracy that ranges from 92% to 97%.

Keywords

Serverless • IoT • Scheduling • Cloud functions • GPU

Introduction

Serverless computing (also known as Functions-as-a-Service (FaaS)) is a popular cloud service for hosting and automatically scaling applications [1-3]. Originally designed for web services, serverless computing defines a simple, event-driven programming model and cloud platform with which developers write simple, short-lived functions that are invoked by the platform in response to specific system-wide events [4, 5]. Server less platforms automatically configure and provision isolated execution environments on-demand and users pay only for the resources their functions use during execution.

Moreover, serverless computing has been extended to work at the "edge" of the network to reduce the response latency and bandwidth associated with public cloud use by data-driven applications [6-8]. Doing so is challenging, however, because computing and storage resources are scarce at the edge, whereas public/private clouds may offer specialized hardware that can significantly speed up machine learning applications.

In this work, we investigate the use of serverless computing across the edge and public cloud deployments. To leverage hardware acceleration and distributed (multi-cloud) scheduling within a server less architecture, we have developed STOIC, a framework for distributing and executing analytics applications across multi-tier IoT (sensing-edge-cloud) settings. Specifically, STOIC optimizes the end-to-end process of packaging, transferring, scheduling, executing, and result retrieval for machine learning applications in these settings.

***Address for Correspondence:** Michael Zhang, Department of Computer Science, University of California, Santa Barbara, United States; E-mail: lebo@cs.ucsb.edu*

Received: August 04, 2021; **Accepted:** August 18, 2021; **Published:** August 25, 2021

STOIC

Architecture

It shows the distributed components of STOIC (Figure 1). At the edge, STOIC gathers application input data, determines whether the lower application latency will be achieved by processing the data on the edge or in the cloud, and then actuates the application's computation using the "best" choice. The public cloud component manages whatever cloud resources are needed to receive the data from the edge, trigger the computation, and return the results to the edge. The edge and cloud systems mirror each other, running Kubernetes overlaid with Kubeless, to provide a uniform infrastructure for the framework [9-11].

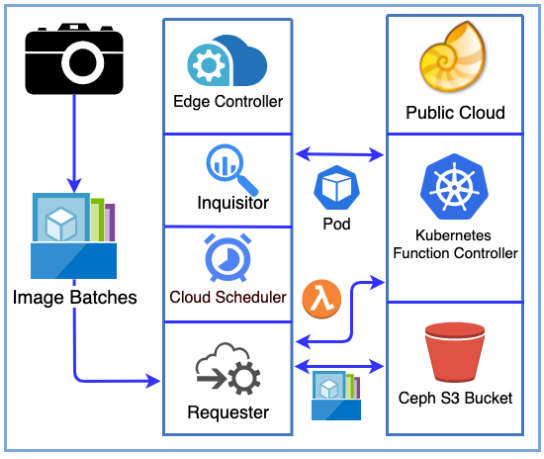

Figure 1. The STOIC architecture.

Our system design is motivated by a need to classify wildlife images in a location where it is possible to site a relatively powerful edge system but where network connectivity is poor. We use STOIC for processing images from multiple, motion-triggered camera traps (sensors) deployed to Sedgwick Natural Reserve currently used to study ecological land use [12].

The Edge Controller is a server that communicates wirelessly with the sensors and triggers analysis and computation upon their arrival. When a camera trap detects motion, it takes photos and persists the images in flash storage buffer, where human experts would label images for training tasks. Periodically, sensors transfer saved photos to the edge controller. STOIC

runs on the edge controller and its executions are triggered by the arrival of image batches. It consists of three major components:

• The cloud scheduler predicts the total latency based on historical measurements for each available runtime.

• The requester takes as input the runtime and cloud predicted by the scheduler to have the least latency. The requester also saves the image package in an object storage service running in this cloud. It then triggers a serverless function (running in a Kubernetes pod) via a HTTP request to process the images.

• The inquisitor monitors public cloud deployment time. To enable this, it periodically in the background deploys each runtime and records the deployment times in a database. We use the inquisitor to establish the historical time series for predicting the deployment latency of remote runtimes.

Public/private cloud: To investigate the use of the server less architecture with hardware acceleration, we employ a shared, multi-university, GPU cloud, called Nautilus, as our remote cloud system [13]. The STOIC cloud/GPU runtimes use Kubernetes and Kubeless for serverless function execution and Ceph for object storage on the public cloud.

A major challenge that we face with such deployments is hardware heterogeneity and performance variability. This heterogeneity impacts application execution time in three significant ways. First, different CPU clock rates affect the transfer of datasets from the main memory to GPU memory. Second, there is significant latency and performance variability between runtimes and the storage service (which hold the datasets and models). Third, the multi-tenancy of nodes allows other jobs to share computational resources with our applications of interest at runtime. These three factors negatively make it difficult for users to determine which runtime to use and when to execute locally. With STOIC, we address these challenges via a novel scheduling system that adapts to this variability.

A major challenge that we face with such deployments is hardware heterogeneity and performance variability. This heterogeneity impacts application execution time in three significant ways. First, different CPU clock rates affect the transfer of datasets from the main memory to GPU memory. Second, there is significant latency and performance variability between runtimes and the storage service (which hold the datasets and models). Third, the multi-tenancy of nodes allows other jobs to share computational resources with our applications of interest at runtime. These three factors negatively make it difficult for users to determine which runtime to use and when to execute locally. With STOIC, we address these challenges via a novel scheduling system that adapts to this variability.

Runtime scenarios: To schedule machine learning tasks across hybrid cloud deployments, we define four runtime scenarios: (A) Edge-A VM instance on the edge cloud with AVX2 support [14]. (B) CPU-A Kubernetes pod on Nautilus containing a single CPU with AVX2 support [14]. (C) GPU1-A Kubernetes pod on Nautilus containing a single GPU. (D) GPU2-A Kubernetes pod on Nautilus containing two GPUs. STOIC considers each of these deployment options as part of its scheduling decisions.

Execution time estimation: The total response time (Ts) includes data transfer define total response time (Ts) as Ts=Tt+Td+Tp.

Transfer time (Tt): Measures the time (Tt), runtime deployment time (Td), and the corresponding processing time (Tp). We time spent in transmitting a compressed batch of images from the edge controller to edge cloud and public cloud.

Runtime deployment time (Td): Measures the time Nautilus uses to deploy the requested kubeless function. Since the scarcity of computation, it is common that multi-GPU runtime takes longer to deploy than single-GPU and CPU runtimes. Note that, for edge runtime, the deployment time zeroes out since STOIC executes the task locally in the edge cloud. To accurately predict deployment time, we analyse deployment times as a time series using three methods:

• Auto-regression modelling,

• Average sliding window, and

• Median sliding window.

It shows representative analytics for GPU1 deployment time, in which MAE oscillates as window size varies (Figure 2). We observe that the median sliding window reaches a lower minimum MAE than the average sliding window at optimal window size. As listed in, all three runtimes achieve the lowest minimum MAE using the median sliding window (Table 1). Therefore, STOIC adopts this methodology for deployment time prediction.

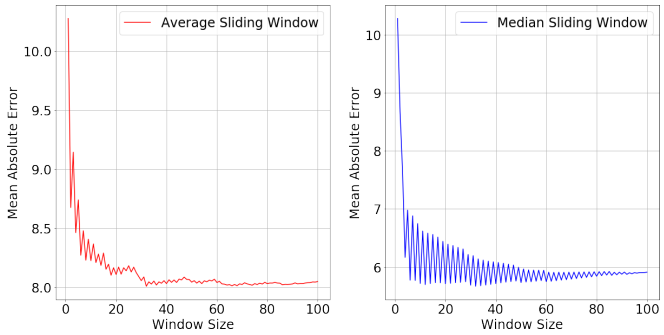

Figure 2. The Mean Absolute Error (MAE) of deployment time for the GPU1 runtime. The x-axis is the window (history) size. The left subplot is MAE when STOIC uses the average sliding window; the right subplot is MAE when STOIC uses the median sliding window.

Modeling	Runtime	Optimal window size	Minimum MAE
Auto Reg	CPU	15	8.977
Auto Reg	GPU1	15	9.605
Auto Reg	GPU2	15	17.918
Avg. SW	CPU	33	7.714
Avg. SW	GPU1	31	8.006
Avg. SW	GPU2	91	16.52
Med. SW	CPU	13	5.96
Med. SW	GPU1	31	5.668
Med. SW	GPU2	27	14.48

Table 1. Mean Absolute Error of three time series modelling methods for runtime deployment time: auto-regression (Auto Reg), average sliding window (Avg. SW), and median sliding window (Med. SW). The median sliding window achieves the lowest minimum MAE at optimal window size (that with the lease MAE) for all three runtimes.

Processing time (Tp): Is the execution time of a specific machine learning task and the target of task scheduling across the hybrid cloud. STOIC formulates a Bayesian Ridge Regression on execution time histories of STOIC jobs, and uses it to predict processing time relative to input (image batch) size [15]. We also augment regression using a random sample consensus (RANSAC) technique, which iteratively removes outliers from the regression [16].

Adaptability: Depicted in, we observe that actual total latency varies significantly and predicted total latency has a non-negligible difference from the actual total latency at the beginning of the experiment (Figure 3). However, over time, as STOIC learns the various latencies of the system, the difference is significantly reduced. In, we report the percentage mean absolute error (PMAE), which we compute as the MAE divided by mean latency (Table 2). The decrease in all three PMAE values in the second half of the execution traces also show STOIC's adaptability.

	Deployment Td	Processing Tp	Total Ts
First half	42.70%	11.20%	15.80%
Second half	29.20%	5.30%	9.20%

Table 2. The percentage mean absolute error (PMAE) of deployment, processing, and total latency.

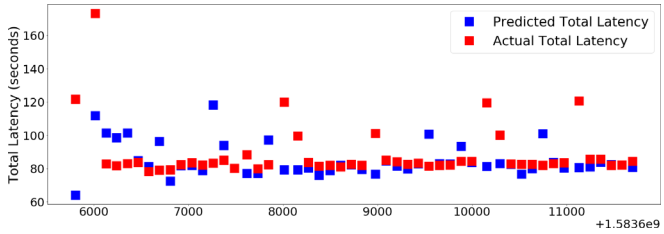

Figure 3. The comparison of predicted and actual total latency on 50 GPU1 benchmark executions with 150-image batch size. The x-axis is the epoch time and the y-axis is the total latency.

Workflow

STOIC considers two workflows upon receiving an image batch: selector mode and duplicator mode. Both are depicted in Figure 4. In selector mode, STOIC selects the runtime with the shortest estimated response time and deploys it locally (Edge) or remotely (non-Edge). To pivot STOIC to variable deployment time, we consider a second workflow called duplicator mode, in which the scheduler selects a public cloud runtime, the requester also deploys the job on the edge cloud. It then terminates edge cloud execution if the remaining time at edge cloud is longer than the expected processing time (Tp) at the GPU runtime once deployment completes. This "lagging decision" mechanism reduces the variability of deployment time in the prediction. As a result, STOIC must only consider processing time, which is more accurately predicted, to deploy tasks.

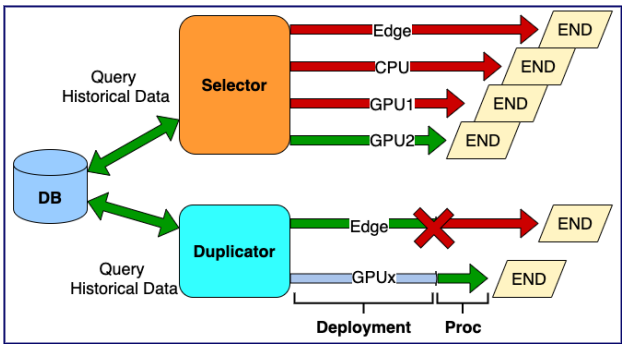

Figure 4. The selector and duplicator modes of STOIC.

Evaluation

Experiment setup

The image processing application that we use as a benchmark classifies animal images from a wildlife monitoring system called "Where's The Bear" (WTB) [17]. "Where's The Bear" is an end-to-end distributed data acquisition and analytics system that automatically analyses camera trap images collected by cameras sited at the Sedgwick Natural Reserve in Santa Barbara County, California [12]. Our deployment includes an edge cloud located near the cameras where it acquires the image data. The edge cloud is connected via a slow (microwave) link to a private cloud located at a research facility located approximately 50 miles from the site. In this work, we explore using the Nautilus distributed GPU cloud as the public cloud, in conjunction with the edge cloud to optimize image classification on a Convolutional Neural Network (CNN) implemented by Tensor flow and Scikit learn [13, 18, 19].

Discussion

In total, there are five classes that we consider: Bird, Fox, Rodent, Human, and Empty, by which we label images for training tasks and evaluate models by inference. Since class size is unbalanced due to the frequency of animal occurrences, we up-sample minority classes (e.g. fox) using the Keras Image Data Generator [20]. Doing so ensures that the classification model is not biased. We resize every image in the image dataset to 1920 1080, and for each class, the dataset contains 251 images used to train the CNN model. Once model training is complete, the application stores this model in hdf5 format in object storage at both edge cloud and Nautilus.

As described previously, STOIC moves images from the wildlife refuge to the public cloud in batches. To better harness the multiple GPU runtime of the public cloud, the application spawns a process (worker) for each GPU and adds all images in a batch to a shared asynchronous queue. Upon the execution, workers remove images (one at a time) from the shared queue until it is exhausted. This mechanism ensures multiple GPU runtimes evenly divide the workloads among GPUs and achieve quasi-linear acceleration at the application level, where the perfect linear speed-up is unattainable because of model loading and memory transfer overhead [21].

To drive this experiment, we use the workload generator to facilitate faster-than-real time evaluation of STOIC. The generator uses an image series and their inter-arrival patterns from a camera trap image corpus ranging from 2013 to 2017. Shows example histograms for processing time, transfer time, and deployment time on Nautilus for GPU1 runtime using 150 batches drawn from the workload generator (Figure 5). On the x-axis, we show the elapsed time for processing time, transfer time, and deployment time respectively. Note that processing time and transfer times are relatively stable compared to deployment time.

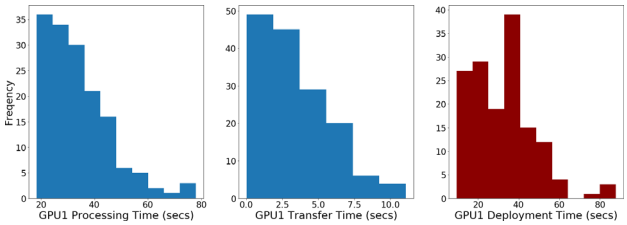

Figure 5. The distribution of three components in total response time (Ts) of 150 executions on GPU1 runtime: Processing time (Tp), Deployment time (Td), and Transfer time (Tt). The x-axis represents the time range, while the y-axis is the frequency of executions. The deployment time, which is depicted in the red histogram, is volatile and error-prone to prediction.

Selector evaluation

We first evaluate STOIC selector mode for a 24-hour period consisting of 162 image batches, the sizes of which are drawn randomly from the work-these four options 92% of the time. That is, STOIC correctly identifies the fastest option with a success rate of 92%. Further, we define MIN-LAT (minimum latency scheduler), which is an oracle scheduler that is 100% correct on selections of runtime. Such scheduler would have resulted in an aggregate total latency of 10022 seconds, whereas the worst case, in which the scheduler selects the highest-latency runtime for every run, has an aggregate latency of 35940 seconds, compared to a STOIC aggregate latency of 10770 seconds. Thus STOIC achieves an aggregate latency 70% (3.33 xs) faster than the worst case.

Duplicator evaluation

It shows the performance of the Duplicator using the edge and one GPU and, separately, the edge and two GPUs from Nautilus (Table 3). Duplicator of gpu1 runtime achieves the highest success rate (97%), while duplicator of gpu2 has the lowest average latency.

	Success rate	Versus MIN-LAT	Versus worst case
Selector	92%	105%	30%
Duplicator edge vs. GPU1	97%	102%	30%
Duplicator edge vs. GPU2	95%	101%	30%

Table 3. The comparison of selector and duplicators.

Conclusion

In this paper, we propose a framework, called STOIC, for executing machine learning applications in IoT-cloud settings using the serverless architecture. We present the design principles, implementation details, the feedback control mechanism, and different modelling methodologies to address the variability in the edge and public cloud deployments. Our empirical evaluation demonstrates STOIC can schedule tasks on local and remote deployments to achieve a speedup of 3.3x versus our baseline scenario. STOIC's success rate for prediction placement ranges from 92% to 97% for the application and datasets that we study.

As part of future work, we plan to investigate substituting RANSAC with Gradient Boosting Regression Trees (GBRT) to capture the non-linearity in the processing time due to heterogeneous hardware across deployment options (runtimes). We also plan to investigate model check-pointing in duplicator mode to better utilize computational resource on edge cloud and to improve the overall performance of the STOIC system.

Acknowledgments

This work has been supported in part by NSF (CNS-1703560, CCF-1539586, ACI-1541215), ONR NEEC (N00174-16-C-0020), and the AWS Cloud Credits for Research program.

This work was performed in part at the University of California Natural Reserve System Sedgwick Reserve DOI: 10.21973/N3C08R.

References

1. AWS Lambda. AWS. (2020).
2. The Serverless Applications Framework. Serverless. (2020).
3. Azure Functions. Microsoft Azure. (2020).
4. Build a Serverless Web Application. AWS. (2019).
5. AWS Lambda for Microservices. AWS. (2019).
6. AWS IoT Greengrass. AWS. (2019).
7. Azure IoT Hub. Microsoft Azure.
8. Azure IoT Edge. Microsoft Azure.
9. API Reference. Kubernetes. (2020).
10. Brendan, Burns, Grant Brian, Oppenheimer David and Brewer Eric, et al. "Borg, Omega, and Kubernetes System Evolution." *Syst Evol* 14 (2016): 1-24.
11. Kubeless. GitHub. (2020).
12. Sedgwick Natural Reserve. Sedgwick Reserve. (2020).
13. Nautilus Documentation. Nautilus. (2020).
14. Documentation. Oracle. (2020).
15. Bayesian Ridge Regression. Scikit. (2020).
16. Random Sample Consensus. RANSAC. (2020).
17. Elias, Rosales Andy, Nevena Golubovic, Chandra Krintz and Wolski Rich. "Where's the bear?-Automating Wildlife Image Processing Using IoT and Edge Cloud Systems." *IEEE ACM Second Int Conf Internet Things Des Implement* 1 (2017): 247-258.
18. Yann, LeCun and Yoshua Bengio. "Convolutional Networks for Images, Speech, and Time Series." *Handbook Brain Theory Neural Netw* 1 (1998): 255-258.
19. Pedregosa, Fabian, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, et al. "Scikit-Learn: Machine Learning in Python." *J Mach Learn Res* 12 (2011): 2825-2830.
20. Image Data Preprocessing. Keras. (2020).
21. Campos, Victor, Francesc Sastre, Maurici Yagues and Jordi Torres, et al. "Scaling a Convolutional Neural Network for Classification of Adjective Noun Pairs with TensorFlow on GPU Clusters." *IEEE ACM Int Symp Cluster Cloud Grid Comput* 1 (2017):677-682.

How to cite this article: Zhang, Michael, Chandra Krintz and Rich Wolski."Function-As-A-Service Acceleration for IoT Applications on Hybrid Cloud." *J Sens Netw Data Commun* 10 (2021): 133.