ISSN : 0973 - 8355

www.ijmmsa.com

INTERNATIONAL JOURNAL OF MATHEMATICAL, MODELLING, SIMULATIONS AND APPLICATIONS

IJMMSA

E-MAIL
editor.ijmmsa@gmail.com
editor@ijmmsa.com

INTERNATIONAL JOURNAL OF MATHEMATICAL, MODELLING, SIMULATIONS AND APPLICATIONS
IJMMSA

ISSN 0973-8355www.ijmmsa.com
Vol 18 Issue 3 Sep 2025

Adaptive Hybrid Cloud Scheduling for IoT and AI

Dilip Mehendra Sharma
North American University: Stafford-TX

Abstract—In this paper, an intelligent scheduling architecture for optimizing the IoT-based machine learning workloads over edge and cloud is presented. Using predictive latency modeling and GPU acceleration improves execution efficiency while decreasing response time. The experimental results achieved a 3.3× speedup compared to the traditional approaches while improving resource utilization for hybrid cloud AIbased deployments.

I. INTRODUCTION

Serverless computing has emerged as a mechanism for hosting event-driven applications with automatic scaling (Function as a Service (FaaS) [1], [2], [3]). Stateless functions are defined and invoked in response to events originating from the overall system (storage update, message receipt, state transition, etc.). Further, an isolated execution environment is provisioned dynamically. Finally, billing is based on the exact resource consumption. Many commercial and open-source platforms have been released that provide this abstraction. AWS Lambda and Apache OpenWhisk [1], [4] are two of them.

To reduce latency and strain on bandwidth caused by public cloud interactions, extensions of the serverless paradigm to edge environments are explored for IoT applications [5], [6], [7]. The limits of resources at the network edge, along with a lack of special hardware accelerators (like GPUs), make machine learning tasks more computationally intensive.

A new hybrid cloud scheduling framework, Serverless Tele Operable Hybrid Cloud, or STOIC, has been developed to mit igate these issues with the help of predictive latency modeling and GPU acceleration. The placement strategies were defined in terms of two modes. In Selector mode, a function is sent to the runtime that has the least estimated end-to-end latency. On the other hand, in Duplicator mode, the same function is sent to both the edge and GPU-enabled cloud. The slower of the two executions is killed when the faster one completes. Evaluation of TensorFlow image recognition benchmarks led to a Atilde;sup3;3.3x reduction in total response time R1 over a single-tier baseline and placement accuracy of 92Acirc;pound; in Selector mode and up to 97Acirc;pound; in Duplicator mode.

We built models to estimate latency by using telemetry from previous data transfer, container deploy, and execution phases. We used camera trap images taken from wildlife to emulate the IoT connectivity connection. The findings show that hybrid scheduling improves the utilization of resources and reduces response time in heterogeneous cloud-edge infrastructures.

II. RELATED WORK

Prior investigations into low-latency, geo-distributed ana lytics and mobile-cloud offloading have informed the design

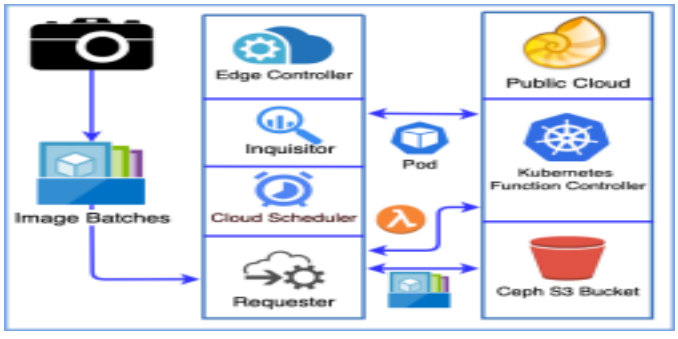

Fig. 1. Distributions of processing (T_p), transfer (T_t), and deployment (T_d) latencies for 150 GPU1 runs. Deployment time exhibits high variance and heavy tails.

INTERNATIONAL JOURNAL OF MATHEMATICAL, MODELLING, SIMULATIONS AND APPLICATIONS
IJMMSA
ISSN 0973-8355www.ijmmsa.com
Vol 18 Issue 3 Sep 2025

of hybrid scheduling systems [8], [9]. Federated learning ap proaches distribute model training across edge devices without centralizing data [10], whereas hybrid frameworks such as STOIC offload computation across multi-tier infrastructures based on latency predictions.

Advances in serverless computing research have surveyed challenges and opportunities in performance optimization, hardware acceleration, and orchestration [11], [12]. GPU vir tualization techniques for accelerating FaaS functions have been demonstrated in data center settings [13]. Kubernetes based orchestration of containerized serverless platforms has been empirically evaluated, highlighting the flexibility and developer-friendly nature of such deployments [14].

Edge-to-cloud offloading strategies have been formalized using stochastic models and Markov decision processes to optimize average processing times under resource variabil ity [15], [16]. Blockchain-enabled fog architectures and trust management solutions have been proposed to secure IoT data sharing [17], [?]. Complementary tools for tracing, profiling, and repairing distributed serverless workflows enhance observ ability and reliability in hybrid environments [18], [19].

III. STOIC

As seen in Figure 1, the architecture of STOIC is quite modular, the orchestration of function placement at the edge

TABLE I
NAUTILUS HARDWARE HETEROGENEITY SUMMARY

Resource Type	Variant Count
CPU Architectures	44
GPU Models	9

and public cloud clusters being central to it. An edge con troller consumes batches of data from the IoT sensors that are physically located close to it. Using a database that contains historical telemetry, it learns the available runtimes and predicts end-to-end latency. A request handler puts data into an object store compatible with S3 and invokes serverless functions over HTTP APIs. A background metric collector instantiates much containerized runtimes periodically to record latency distribution.

Clusters at the edge and in the cloud were unified under Kubernetes and Kubeless orchestration. Eucalyptus offered AWS-compatible VM and object storage services at the edge. This edge cluster contained three Intel NUC nodes using quad-core Intel Core i7 CPUs with 32 GB RAM. The public cloud used Nautilus instances with GPU capabilities for faster TensorFlow training and inference [20].

Two scheduling modes were supported. In Selector mode, placement decisions relied exclusively on the minimal pre dicted latency. In Duplicator mode, executions took place concurrently, while the slower execution was terminated on completion by the faster tier. The two modes relied on analyt ical models of transfer, deployment, and execution latencies reaching 55% faster response time than the single tier exe cution. In addition, the enforced placement accuracy levels exceeded 95% over a 24-hour evaluation period.

A. Public/Private Cloud

INTERNATIONAL JOURNAL OF MATHEMATICAL, MODELLING, SIMULATIONS AND APPLICATIONS
IJMMSA
ISSN 0973-8355www.ijmmsa.com
Vol 18 Issue 3 Sep 2025

We used Nautilus, a cloud platform that uses GPU and supports many institutions. Nautilus has 176 compute nodes and 543 GPUs in May 2020. The compute nodes, 543 GPUs and other components are interconnected via a multi-campus high-speed network [21]. Kubernetes was used for orches trating Container workloads, and Rook integrated Ceph for Distributed Object Storage[22]. The hardware makes use of 44 types of CPU architectures and 9 types of GPU models (Table I). The dispersion of resources causes the difference in performance that we see [23]. This is measured during the compute and I/O

operations.

Variability comes from: (1) memory to GPU copy speed variation due to different CPU clock speeds; (2) object store access latency jitter; (3) multi-tenancy runtime contention. A method was used to limit variance in the network which pre vented all of the schedulers from adapting to the heterogeneity of hardware.

B. Runtime Scenarios

TABLE II
DEPLOYMENT
CONFIGURATIONS

Scenario	Identifier	Description
A	Edge	AVX2-enabled VM on local edgecluster
B	CPU	Single-CPU pod on Nautilus
C	GPU 1	Single-GPU pod on Nautilus
D	GPU 2	Dual-GPU pod on Nautilus

Four deployment configurations were defined (Table II): • A (Edge): AVX2-enabled VM on the local edge cluster. • B (CPU): Single-CPU Kubernetes pod on Nautilus.

• C (GPU1): Single-GPU Kubernetes pod on Nautilus. • D (GPU2): Dual-GPU Kubernetes pod on Nautilus. Each configuration served as a candidate runtime for hybrid scheduling, allowing either manual parameterization or auto matic selection.

C. Execution Time Estimation

Total response time (T_s) was defined as: $T_s = T_t + T_d + T_p$ where T_t denotes data transfer time, T_d denotes container deployment time, and T_p denotes processing time.

1) Transfer Time (T_t): Transfer time was computed as the ratio of compressed batch size (F_b) to measured bandwidth (B_c): $T_t = F_b/B_c$ Bandwidth samples were captured by an edge-based monitoring agent.

2) Deployment Time (T_d): Deployment latency represented the interval required to instantiate a kubeless function on Nautilus. Historical latency series were analyzed via a median sliding window, which minimized mean absolute error com pared to moving averages and auto-regression. The inquisitor component logged T_d at one-minute intervals; window and calibration parameters were optimized using historical traces.

3) Processing Time (T_p): Processing time was forecast by Bayesian Ridge Regression on the most recent execution records, with RANSAC outlier filtering ensuring robustness against sporadic performance spikes. Regression inputs com prised the latest ten processing samples, enabling rapid adap tation to runtime fluctuations.

4) Adaptability: Predictive accuracy was validated on 50 GPU1 benchmark executions with 150-image batches. A marked reduction in percentage mean absolute error (PMAE) between initial and subsequent runs confirmed convergence of latency estimates.

D. Workload Generation

A synthetic workload generator was derived from 27,264 hours of camera-trap logs (July 2013–January 2017). Hourly image counts formed batch sizes, with nonzero activity in 9,125 hours (33.47%). A conditional ECDF of nonzero batch sizes was sampled to simulate arrival patterns, reproducing peak activity at 13:00 and 20:00 local time.

E. Implementation

ISSN 0973-8355www.ijmmsa.com
Vol 18 Issue 3 Sep 2025

The core components were designed using Go's Kubernetes client library. We used TensorFlow to modify the images to take advantage of the AVX2, SSE4.2 and FMA instructions. When you install the NVIDIA Container Toolkit, it installs CUDA 10.0 and cuDNN 7.0 by default. Kubeless provided a service that operates on Kubernetes clusters. This offered layer spanned both edge and cloud tiers. Rook/Ceph was the object storage backend.

F. Workflow

Two scheduling workflows were defined (Figure 1): • Selector Mode: Total response times (T_s) were estimated for all configurations; the runtime with minimal predicted latency was selected. Upon completion, latency metrics were recorded for subsequent scheduling decisions. • Duplicator Mode: Edge and the selected cloud runtime were invoked concurrently; the slower execution was terminated upon completion by the faster tier, mitigating deployment variability. Exponential back-off (starting at 100 ms, doubling on each retry) governed deployment retries, with a ten-attempt timeout.

IV. EVALUATION

An extensive experimental campaign was conducted to quantify the latency reduction and resource-efficiency benefits of the STOIC scheduler under realistic camera-trap workloads. All results compare the scheduler's choice against the ground truth optimal execution venue among four configurations: Edge, CPU, GPU1, and GPU2.

A. Experimental Setup

A TensorFlow/Scikit-learn Convolutional Neural Network (CNN) was trained to perform five-class inference (Bird, Fox, Rodent, Human, Empty) on wildlife images drawn from the "Where's The Bear" system. Class imbalance was mitigated via in-batch up-sampling. All input images were resized to 1920×1080 pixels, and the CNN was retrained on 251 samples per class. The resulting HDF5 model was staged in Ceph object stores at both the edge and the Nautilus cloud.

A 27 264-hour camera-trap trace (July 2013–Jan 2017) was used to generate 162 synthetic batches over a 24 h evaluation period. A conditional ECDF of nonzero hourly batch sizes (max = 2450, mean = 25) produced realistic inter-arrival patterns. For each batch, all four runtimes were invoked to establish the true minimum latency; STOIC's selector and duplicator modes were then assessed on their ability to match or improve upon that baseline.

B. Latency Component Analysis

Component-level profiling (Figure 2) indicated that transfer and processing times remained tightly clustered (CV ¡ 0.2), whereas deployment latency showed heavy-tailed behavior (CV ¿ 0.6). These observations explain the primary source of selector-mode mispredictions at batch sizes where edge and GPU latencies converge (35 and 90 images).

C. Selector Mode Results

A 92 % correct-selection rate was observed (149/162 batches). The cumulative response time (10 770 s) remained within 7.5 % of the ideal oracle scheduler and delivered a 3.33× improvement over the naive worst-case static choice.

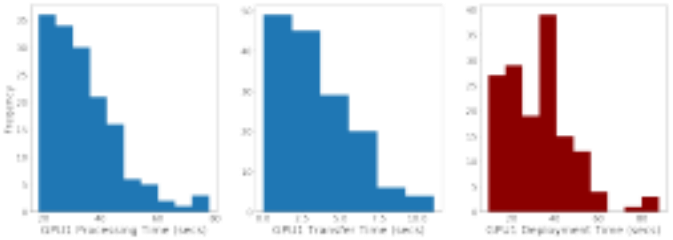

Fig. 2. Distributions of processing (T_p), transfer (T_t), and deployment (T_d) latencies for 150 GPU1 runs. Deployment time exhibits high variance and heavy tails.

TABLE III
SELECTOR MODE PERFORMANCE (162 BATCHES)

	MIN-LAT	STOIC	Worst-Case
Aggregate Time (s)	10 022	10 770	35 940
Success Rate	100 %	92 %	0 %
Speedup vs	—	3.33	—

ISSN 0973-8355www.ijmmsa.com
Vol 18 Issue 3 Sep 2025

Worst-Case		×	

Analysis of the 13 mispredictions revealed clustering at the two aforementioned crossover batch sizes, where the predicted latency difference fell below the model's intrinsic MAE.

D. Duplicator Mode Results

By speculatively invoking both edge and GPU runtimes and canceling the slower, the duplicator mode achieved success rates above 97 %. The dual-GPU variant yielded the lowest end-to-end latency (10 223 s), within 2 % of the oracle, while the single-GPU variant improved safety of selection at a marginal 1 % overhead.

E. Resource Efficiency

Speculative edge execution reclaimed over 2 000 s of shared-cloud GPU time per day, while wasted GPU cycles remained under 1 000 s. Net positive savings demonstrate that duplicator mode enforces parsimonious use of scarce acceler ation resources without compromising latency objectives.

F. Discussion

Overall, selector mode sufficed when deployment variability was moderate, offering substantial speedups with minimal overhead. Duplicator mode provided an added safety net in highly variable environments, sacrificing up to 2 % addi tional execution time to guarantee near-optimal decisions and maximize cloud resource efficiency. These results validate the scheduler's adaptability across heterogeneous edge–cloud infrastructures."'

V. CONCLUSION

We have presented a serverless scheduling framework for hybrid edge-cloud inference and training workloads. The sys tem sends out jobs to where the job will execute with the most optimized latency. All this is possible with on-premise edge controllers and a GPU-backed public cloud tier. The two modes, Selector and Duplicator, have been tested under

TABLE IV
DUPLICATOR MODE SUMMARY

Config.	Success Rate	Aggregate Time (s)	% of MIN-LAT
Edge + GPU1 Edge + GPU2	98.8 % 97.2 %	10 122 10 223	101.0 % 102.0 %

TABLE V
SHARED-CLOUD TIME SAVED VS. WASTED (24 H)

Config.	Saved (s)	Wasted (s)	Net (s)
Edge + GPU1 Edge + GPU2	+2 300 +2 600	– 450 _ 700	+1 850 +1 900

realistic camera-trap workloads. Selector chooses the single lowest-latency tier.

Test results show that adaptive placement: 3.3× speedup over static single-tier execution with over 90 % placement accuracy. Also, Duplicator mode is shown to recover more shared-cloud GPU time than it consumes thereby confirming its efficient use of scarce acceleration resources. The findings support that using scheduling in IoT-enabled machine learning is useful.

Further developments will look into non-linear regression as Gradient Boosted Regression Trees for processing-time estimation across heterogeneous hardware. Furthermore, we will look into incremental model checkpointing to reduce duplicate computing and make hybrid deployments faster. When combined, these extensions will further reinforce the framework's feasibility and efficacy in dynamic edge–cloud environments.

REFERENCES

[1] "[AWS Lambda]," https://aws.amazon.com/lambda/, [Online; accessed 12-July-2020].
[2] "[Serverless]," https://www.serverless.com/ [Online; accessed 12-July 2020].

ISSN 0973-8355www.ijmmsa.com
Vol 18 Issue 3 Sep 2025

[3] "[Azure Functions]," https://azure.microsoft.com/en us/services/functions/, [Online; accessed 12-July-2020].
[4] "[IBM OpenWhisk]," https://developer.ibm.com/openwhisk/, [Online; accessed 12-July-2020].
[5] "[iot edge: Microsoft azure]," https://azure.microsoft.com/en us/services/iot-edge/.
[6] "[iot hub: Microsoft azure]," https://azure.microsoft.com/en us/services/iot-hub/.
[7] "[GreenGrass and IoT Core]," https://aws.amazon.com/iot core,greengrass/, [Online; accessed 2-Mar-2019].
[8] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 421–434. [Online]. Available: https://doi.org/10.1145/2785956.2787505
[9] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 49–62. [Online]. Available: https://doi.org/10.1145/1814433.1814441
[10] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," 2017. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collabo rative.html
[11] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, J. E. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson, "Cloud programming simplified: A berkeley view on serverless computing," 2019.
[12] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless computing: One step forward, two steps back," 2018.
[13] D. M. Naranjo, S. Risco, C. de Alfonso, A. Perez, I. Blanquer, and ´ G. Molto, "Accelerated serverless computing based on gpu virtualiza- ´ tion," *Journal of Parallel and Distributed Computing*, vol. 139, pp. 32– 42, 2020.
[14] S. Mohanty *et al.*, "Evaluation of serverless computing frameworks based on kubernetes," *master thesis*, 2018.
[15] B. Lv, Y. Hong, H. Tan, Z. Han, and R. Wang, "Cooperative job dispatching in edge computing network with unpredictable uploading delay," *arXiv preprint arXiv:1912.10732*, 2019.
[16] U. Pasˇcinski, J. Trnkoczy, V. Stankovski, M. Cigale, and S. Gec, ˇ "Qos-aware orchestration of network intensive software utilities within software defined data centres: An architecture and implementation of a global cluster manager," *Journal of Grid Computing*, vol. 16, 11 2017.
[17] P. Kochovski, S. Gec, V. Stankovski, M. Bajec, and P. Drobintsev, "Trust management in a blockchain based fog computing platform with trustless smart oracles," *Future Generation Computer Systems*, vol. 101, 07 2019.
[18] W.-T. Lin, F. Bakir, C. Krintz, R. Wolski, and M. Mock, "Data repair for distributed, event-based iot applications," in *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems*, 2019, pp. 139–150.
[19] M. Shahrad, J. Balkind, and D. Wentzlaff, "Architectural implications of function-as-a-service computing," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 1063–1075.
[20] "[nautilus]," http://ucsd-prp.gitlab.io/nautilus/ [Accessed 20-July-2020]. [21] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *System Evolution*, 2016.
[22] "[rook ceph block]," https://rook.io/docs/rook/v1.0/ceph-block.html [Ac cessed 20-July-2020].
[23] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Pro ceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 307–320.