Intelligent Optimization of Distributed Pipeline Execution in Serverless Platforms: A Predictive Model Approach

Usama Benabdelkrim-Zakan
Alterna Tecnologías S.L
Tarragona, Spain
usama.benabdelkrim@urv.cat

Germán Telmo Eizaguirre-Suárez
Universitat Rovira i Virgili
Tarragona, Spain
germantelmo.eizaguirre@urv.cat

Pedro García-López
Universitat Rovira i Virgili
Tarragona, Spain
pedro.garcia@urv.cat

ABSTRACT

Achieving efficient execution of distributed pipelines in serverless environments is essential to minimize both execution time and operational costs in cloud settings. This paper presents an approach to predict and optimize the duration of a serverless pipeline executed and parallelized with *Lithops*, using a geospatial water consumption analysis pipeline as a case study. The hyperparameters of the *XGBoost* model were optimized using *Optuna*, resulting in a 75.34% reduction in Mean Absolute Error (MAE) compared to a baseline model, and a 79.9% reduction in execution time compared to suboptimal configurations. Additionally, the model reduced the number of necessary pipeline executions by 30% compared to a full Design Space Analysis (DSA), leading to a 30% cost savings. These results highlight the model's ability to significantly improve both execution efficiency and cost-effectiveness, showcasing the benefits of using *Lithops* for serverless pipeline optimization.

CCS CONCEPTS

• **Software and its engineering** → *Software design engineering*; *Operational analysis*; Use cases; *Software design techniques*; • **Computing methodologies** → **Classification and regression trees**; *Feature selection*; *Regularization*; *Cross-validation*; **Supervised learning**; **MapReduce algorithms**.

KEYWORDS

Serverless Computing, Distributed Pipelines, Cloud Computing, Resource Optimization, Machine Learning, XGBoost, Performance Optimization, Geospatial Analysis, Pipeline Configuration, FaaS

ACM Reference Format:
Usama Benabdelkrim-Zakan, Germán Telmo Eizaguirre-Suárez, and Pedro García-López. 2024. Intelligent Optimization of Distributed Pipeline Execution in Serverless Platforms: A Predictive Model Approach. In *10th International Workshop on Serverless Computing (WoSC10 '24), December 2–6, 2024, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3702634.3702951

1 INTRODUCTION

Serverless data analytics pipelines are increasingly popular for processing large datasets due to their scalability, cost-effectiveness, and ease of use. By abstracting the infrastructure management, they allow developers to focus solely on writing code. *Lithops* [1], an evolution of *Pywren* [2], is designed to execute Python functions as serverless tasks across multiple cloud providers, automating resource provisioning and scaling. Through rounds of *map* functions that interact with object storage, these platforms streamline the development of data-intensive applications requiring dynamic scaling and parallelism.

Lithops manages infrastructure across cloud providers to execute functions in parallel, breaking tasks into smaller units and automating resource provisioning based on user configurations like runtime memory and ephemeral storage. Using AWS Lambda's [3] dynamic allocation of vCPUs and memory, it scales resources on demand, optimizing costs for large datasets.

The pipeline executed on Lithops and AWS Lambda processes geospatial water consumption data in multiple stages. It starts by splitting input data into smaller chunks, which are processed in parallel by AWS Lambda functions to handle tasks such as data preparation, raster interpolation, and evaporation computation. Lithops manages task distribution, ensuring efficient use of resources as each Lambda function independently processes its chunk. This approach leverages serverless scalability, enabling seamless handling of complex, large datasets by automatically scaling resources based on demand and reducing operational overhead through pay-per-use billing.

The predicted optimal configuration adjusts key parameters like memory, splits, and vCPUs. For example, allocating more memory allows fewer, larger tasks, reducing cold start overhead, while fewer splits minimize coordination costs. These adjustments improve processing efficiency and cost-effectiveness on serverless platforms.

Despite these advantages, optimizing the performance and cost-efficiency of serverless data analytics pipelines remains a challenge. Execution time and costs depend on various configuration parameters, such as memory, vCPUs, ephemeral storage, and the degree of parallelism (e.g., splits). Finding the optimal configuration is complex due to the multitude of possible parameter combinations. Exhaustive testing through *Design Space Analysis* (DSA) can be costly and inefficient, as it requires multiple pipeline executions under varying settings.

1.1 Problem Statement

The challenge is to find optimal configurations that minimize time and costs without exhaustive testing. For pipelines that run repeatedly, this optimization can lead to significant savings. Complex parameter interactions make manual tuning inefficient, highlighting the need for a smarter solution.

WoSC10 '24, December 2–6, 2024, Hong Kong, Hong Kong

ACM ISBN 979-8-4007-1336-1/24/12
https://doi.org/10.1145/3702634.3702951

1.2 Proposed Solution

We propose a machine learning approach to predict pipeline execution time based on key parameters, enabling efficient identification of optimal settings that minimize both time and cost. By training on a limited set of executions, our model can estimate performance for unseen configurations, making it universally applicable to pipelines on serverless platforms like *Lithops*.

Leveraging the elasticity and pay-per-use model of serverless architecture, our approach ensures automatic scaling of resources, optimizing costs and performance. Predictions on memory, vCPUs, and ephemeral storage help balance speed and cost, allowing users to pay only for what they consume.

We validated this method using a water consumption pipeline in Murcia, Spain [4], processing geospatial climate data through stages like:

(1) Data Preparation: Uploading and converting Digital Terrain Models (DTMs).
(2) Raster Data Interpolation: Parallel interpolation of climate variables.
(3) Computation of Potential Evaporation: Estimating evapotranspiration.
(4) Result Visualization: Generating visual representations.

Though tested with Lithops on AWS Lambda, the approach adapts to other FaaS platforms like Azure Functions and Google Cloud Functions [5, 6], ensuring seamless integration across providers.

1.3 State of the Art

Previous studies have explored resource optimization in serverless environments. Arjona *et al.* [7] developed *Dataplug*, which enhances performance through *Design Space Analysis* (DSA) by varying chunk sizes. However, their approach focuses solely on chunk size, without considering other parameters such as memory or vCPUs. In contrast, our method achieves a 30% cost reduction compared to DSA by optimizing multiple parameters simultaneously.

The Sizeless model [8] predicts optimal memory sizes based on monitoring data from a single configuration, yielding a 2.6% cost reduction and a 16.7% improvement in execution time. Our approach provides greater benefits, achieving a 19.71% cost reduction and up to a 79.9% reduction in execution time.

Overall, our *XGBoost* model optimizes several parameters concurrently, offering substantial time and cost savings compared to prior methods.

1.4 Contributions

We present a machine learning model that predicts pipeline execution time, reducing it by up to 79.9% and lowering costs by 30% compared to DSA. Our model optimizes multiple parameters simultaneously, offering a versatile solution applicable to any serverless data analytics pipeline.

2 METHODOLOGY

Our methodology includes configuration analysis via *Design Space Analysis* (DSA), data preprocessing, hyperparameter optimization, and model evaluation. Lithops automates configuration management and execution on AWS Lambda for efficient parallel processing, utilizing libraries like `numpy`, `pandas`, `rasterio`, `shapely`, `scikit-learn`, and `Optuna` [9–14] for data management and analysis.

2.1 Configuration Analysis Using DSA

The *Design Space Analysis* (DSA) involved executing the pipeline with 148 configurations on Lithops, using AWS Lambda as the underlying serverless platform. This approach allowed for the analysis of key parameters affecting execution time, combining systematic variation with selective random sampling to capture meaningful insights.

Selection of Configurations. Configurations were selected to explore combinations of parameters that significantly impact pipeline performance. A selective approach combined targeted ranges with random sampling, capturing diverse scenarios while reducing the total number of tests. This strategy effectively represented critical variations based on prior experience and realistic use cases.

Parameters Adjusted During the DSA.

- Splits: Range of 2 to 6, to balance parallelism. More than 6 adds unnecessary overhead, while fewer than 2 limits efficiency.
- Allocated Memory: 1,024 MB to 3,008 MB (maximum in Lithops). Less than 1,024 MB was insufficient for the data used, leading to suboptimal performance.
- Ephemeral Storage: 512 MB to 8,192 MB, varied to support different temporary storage needs.
- vCPUs: Indirectly set by allocated memory in AWS Lambda (0.85 to 1.61 vCPUs). Not directly controlled, but adjusted automatically based on memory.
- Input Files and Size: Configurations with 5 or 15 files, ranging from 0.25 to 1 GB, based on a real-world use case analyzing water consumption in Murcia.

Each execution recorded detailed information about the pipeline configuration and the resulting execution time. The key configuration parameters collected are summarized in Table 1.

Table 1: Input Parameters Collected During DSA

Parameter	Description
num_files	Number of input files processed
splits	Number of splits (chunks) used for parallel processing
input_size_gb	Total size of the input data in gigabytes
runtime_memory_mb	Amount of memory allocated for the runtime (MB)
ephemeral_storage_mb	Temporary storage allocated for intermediate data (MB)
worker_processes	Number of worker processes running in parallel
invoke_pool_threads	Number of threads per invocation
vcpus	Number of virtual CPUs allocated

Serverless architecture facilitates dynamic resource allocation, allowing multiple functions to execute concurrently. During execution, resources such as memory and vCPUs are distributed across parallel tasks, managed automatically by the backend (e.g., AWS Lambda). The parameters selected in the DSA directly influence this management; for instance, increasing memory or vCPUs can speed up function execution, while the number of splits controls how workloads are divided. The system optimizes this distribution, ensuring efficient parallel processing without manual intervention, leading to better performance and cost savings.

We collected this comprehensive dataset during the DSA, capturing a wide range of configurations and execution times. This

dataset forms the basis for training and evaluating our predictive model.

2.2 Data Collection and Preprocessing for the Model

The data from the 148 pipeline executions during the DSA were used to train the predictive model. This dataset includes both the original input parameters (Table 1) and additional features created through feature engineering (Table 2), which capture complex relationships between the original parameters.

Table 2: Derived Parameters from Feature Engineering

Derived Parameter	Description
memory_per_file	Memory allocated per file processed (MB)
storage_per_file	Temporary storage per file (MB)
vcpus_per_file	vCPUs allocated per file
files_per_vcpu	Number of files processed per vCPU
size_per_file	Size of each file (GB)
memory_per_gb	Memory allocated per GB of input size
vcpus_per_gb	vCPUs allocated per GB of input size
storage_per_gb	Temporary storage per GB of input size (MB)
threads_per_worker	Threads running per worker process
memory_per_thread	Memory allocated per thread (MB)
vcpus_per_thread	vCPUs allocated per thread
memory_per_thread_vcpus_ratio	Ratio of memory to vCPUs per thread

Before training, several preprocessing steps were applied:

- Data Splitting: Each execution corresponds to a single row in the dataset, and the data was randomly divided into a training set (70%) and a test set (30%).
- Outlier Handling: All data points were retained to help the model learn from inefficient configurations with longer execution times.
- Feature Scaling: Numerical features were scaled to prevent bias from large-scale features.
- Logarithmic Transformation: Applied to the target variable (execution time) to reduce skewness and stabilize variations.
- Data Augmentation: Gaussian noise was added to the training features to improve generalization.

These preprocessing techniques, along with feature engineering, ensured that the model was trained on a diverse set of configurations, capturing complex interactions and improving its ability to predict execution time.

2.3 Hyperparameter Optimization and Model Training

XGBoost was chosen for its efficiency in handling structured data and its ability to capture complex relationships with minimal overfitting. To enhance its performance, we used *Optuna* to conduct a Bayesian hyperparameter search, optimizing:

- Tree depth (*max depth*)
- Learning rate (*learning rate*)
- Number of estimators (*n estimators*)
- Subsampling (*subsample*)
- Feature fraction per tree (*colsample bytree*)
- L1 and L2 regularization (*reg alpha* and *reg lambda*)

The best hyperparameters found are summarized in Table 3.

Using these hyperparameters, the model was trained on 70% of the dataset. The model was evaluated on the remaining 30% using metrics such as *MAE*, *MAPE*, and R^2. A logarithmic transformation on the target variable (execution time) further improved stability in predictions.

Table 3: Best Hyperparameters Found Using Optuna.

Hyperparameter	Value
Max Depth	4
Learning Rate	0.005193
Number of Estimators	2268
Subsample	0.7467
Colsample by Tree	0.9654
Gamma	0.0101
L1 Regularization (Reg_Alpha)	0.0914
L2 Regularization (Reg_Lambda)	0.1549

Hyperparameter tuning and training ensured XGBoost's accuracy with balanced performance.

2.4 Predicting Optimal Configurations

Using the model trained on data from the DSA (Sections 2.1 and 2.2), we aim to optimize pipeline execution time based on specific configuration parameters. Input features include original parameters (Table 1) and derived features (Table 2).

The term 'splits,' also known as 'chunk size,' refers to data division for parallel processing, where more splits increase parallelism. Our objective is to predict execution times, identifying configurations that minimize time and cost without re-running the pipeline. For example, optimizing a pipeline with 24 input files (20 GB) on *AWS Lambda* with 1 vCPU, we tested splits ranging from 3 to 6, yielding the results in Figure 1.

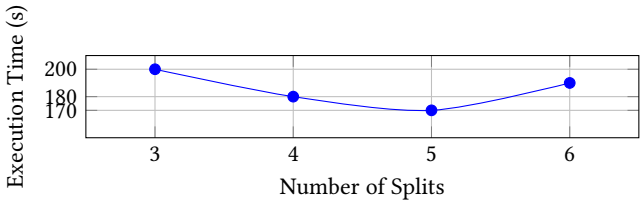

Figure 1: Predicted execution times for different splits.

Results indicate that 5 splits minimize execution time (170 s). Further, as shown in Figure 2, 2,048 MB runtime memory achieves optimal execution time without cost increase, as higher memory does not yield further benefits. Given *AWS Lambda* scaling memory and vCPUs proportionally, memory adjustments directly impact computational capacity.

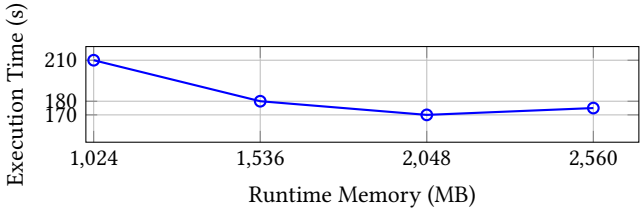

Figure 2: Predicted execution times for different memory allocations.

This approach could be automated through an *Optuna*-like framework, where users define parameter ranges and the framework predicts optimal configurations, streamlining the optimization of pipeline performance while minimizing cost.

2.5 Model Improvement Attempts

Several strategies were explored to enhance the predictive model, but none outperformed the final *XGBoost* approach:

- Synthetic Data (CTGAN): Generated data didn't improve metrics; real data was used.
- Alternative Models: *Random Forest*, *LightGBM* [15], and *MLPRegressor* underperformed, with *XGBoost* as the best model.
- Ensemble Models: Achieved an *MAE* of at least 50, worse than *XGBoost*.
- Feature Selection (RFE): Increased *MAE* to 51.59.

Model	MAE	MAPE	R^2
XGBoost	29.81	8.72%	0.8802
XGBoost with RFE	41.59	11.76%	0.6056
MLPRegressor	44.99	13.53%	0.6126
LightGBM	46.45	11.89%	0.5856

Table 4: Results obtained with different models and techniques

3 RESULTS AND DISCUSSION

This section assesses five approaches to predict and optimize the duration of the geospatial analysis pipeline. These include a machine learning-based model, several baseline methods, and an exhaustive optimization technique.

3.1 Comparative Results Between Approaches

We compared several methods to assess their performance in predicting and optimizing the duration of the geospatial analysis pipeline:

- *XGBoost* Model: Gradient boosting model that captures complex feature interactions.
- Average (Baseline): Predicts using the mean execution time from training data.
- Linear Regression (Baseline): Assumes linear relations between features and duration.
- PCA + Linear Regression (Baseline): Uses PCA for dimensionality reduction before linear regression.
- Design Space Analysis (DSA): Tests configurations exhaustively to find the optimal one but is computationally costly.

Table 5 and Figure 8 present a comparative summary, highlighting *XGBoost*'s strong performance on test data and cross-validation (Avg. MAE).

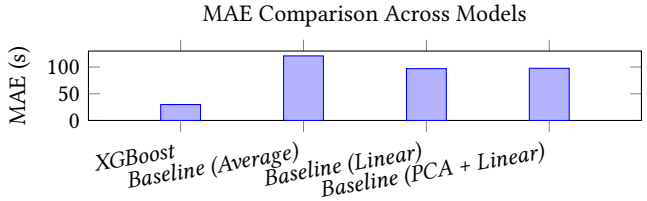

Figure 3: Mean Absolute Error (MAE) comparison across models.

Table 5: Comparison of Models.

Model	MAE (s)	Avg. MAE (CV) (s)	MAPE (%)	R^2
XGBoost	29.81	34.20	8.72%	0.8802
Baseline (Average)	120.90	-	-	-
Linear Regression	97.02	96.62	28.73%	0.3380
PCA + Linear Regression	97.70	92.03	29.04%	0.3240

Table 6: Improvement of *XGBoost* Over Other Models.

Compared Model	Improvement in MAE (%)
Baseline (Average)	75.34%
Linear Regression	69.26%
PCA + Linear Regression	69.47%

The results demonstrate that the *XGBoost* model achieved the highest accuracy, outperforming all baseline models. It reduced MAE by 75.34% compared to the *Baseline (Average)*, and showed improvements of 69.26% over *Linear Regression* and 69.47% over *PCA + Linear Regression* (see Table 6). These results highlight *XGBoost*'s ability to effectively capture complex interactions between pipeline parameters.

The *DSA* approach determines the true optimal configuration through exhaustive testing, while *XGBoost* identifies near-optimal configurations efficiently using only 70% of the data. This predictive capability reduces the necessity for extensive trials, resulting in significant cost savings. The financial benefits of this efficiency are discussed in the following section.

3.2 Cost-Benefit Analysis and Efficiency

Beyond its predictive accuracy, the *XGBoost* model offers significant cost savings compared to the exhaustive *Design Space Analysis* (DSA). Table 7 compares configurations resulting in the minimum and maximum execution durations, along with their respective costs per execution.

Table 7: Configuration Comparison: Minimum vs. Maximum Duration and Costs

Parameter	Minimum Duration	Maximum Duration
Number of Files	5	5
Splits	5	2
Input Size (GB)	0.25	0.25
Runtime Memory (MB)	2000	1024
Ephemeral Storage (MB)	1024	1024
vCPUs	1.13	0.58
Duration (s)	184.08	915.89
Cost per Execution (USD)	0.281	0.350
Cost Difference (USD)	0.069	

The cost difference between these configurations is \$0.069 per execution, resulting in approximately 19.71% savings per run. The initial training cost of \$38.75 for the *XGBoost* model leads to a break-even point after approximately 562 executions, calculated as:

$$N_{\text{break-even}} = \frac{\text{Training Cost}}{\text{Savings per Execution}} = \frac{38.75 \text{ USD}}{0.069 \text{ USD/execution}} \approx 562 \text{ executions} \quad (1)$$

Assuming a rate of 10 executions per day, the model reaches the break-even point in about two months. Figure 4 illustrates the projected cost savings over time.

In summary, the *XGBoost* model delivers substantial cost savings and a rapid return on investment. By efficiently identifying optimal configurations without exhaustive testing, it proves to be an effective and practical solution for ongoing pipeline optimization.

3.3 Comparison of Real vs. Predicted Duration

To evaluate the *XGBoost* model's ability to identify optimal configurations, we tested it on all configurations from the *Design Space*

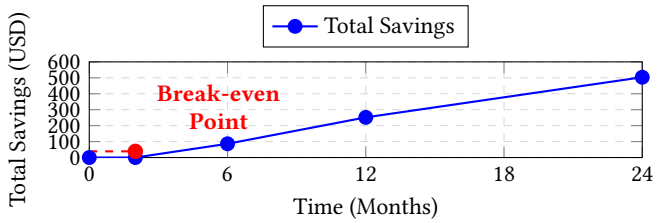

Figure 4: Projected cost savings over time, assuming 10 executions per day. The break-even point occurs at approximately 2 months.

Analysis (DSA), including both seen and unseen setups. The configuration with the shortest real duration, selected by the DSA, was excluded from the training set to evaluate whether the model could identify it as the most efficient in the test set. The results indicated that the configuration with the shortest predicted duration (195.26 seconds) closely matched the actual duration of 184.08 seconds, as presented in Table 7. This alignment demonstrates the model's effectiveness in identifying optimal configurations without the need for exhaustive testing.

3.4 Learning Curve

Figure 5 shows the learning curve of the *XGBoost* model. The graph indicates a significant reduction in error as the size of the training dataset increases. The narrow gap between training and test errors indicates minimal overfitting and strong generalization capacity.

During model training, the *early stopping rounds* technique was implemented to prevent overfitting. This technique stops training when no significant improvements in validation error metrics are observed after a specified number of rounds, ensuring more efficient and robust training.

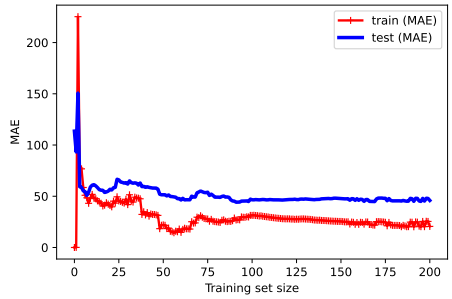

Figure 5: Learning curve of the *XGBoost* model.

3.5 Residual Analysis and Distribution of Relative Errors

The residual analysis, shown in Figure 6, compares the residuals of *XGBoost*, *Linear Regression*, and *PCA + Linear Regression*. The *XGBoost* model displays residuals symmetrically distributed around zero, suggesting greater accuracy and reduced bias. In contrast, the residuals of the other models display greater dispersion, reflecting less reliable predictions.

Figure 7 shows the distributions of actual and predicted values for *XGBoost*. The close alignment suggests the model effectively captures data patterns, though slight discrepancies in lower-value regions indicate potential for improvement. Enhancing results for higher execution times may involve adding more samples from such configurations. Overall, the model demonstrates robust predictive performance, with opportunities for further enhancement through feature engineering or hyperparameter tuning.

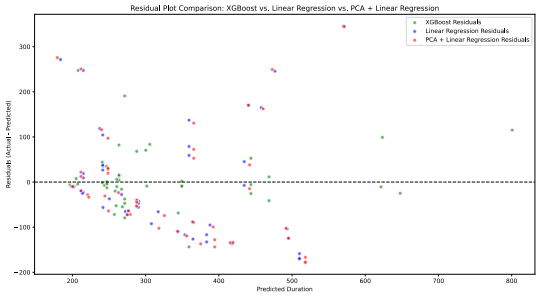

Figure 6: Residual comparison across models: *XGBoost* displays symmetric residuals around zero, suggesting higher accuracy and lower bias.

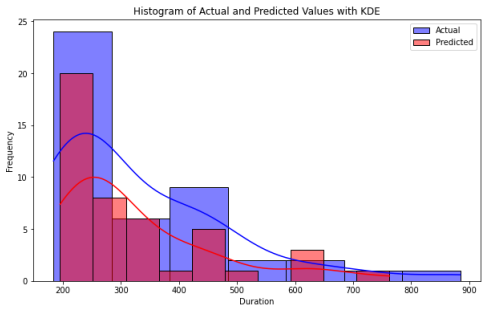

Figure 7: Distribution of relative errors in the test set for the *XGBoost* model.

3.6 Actual vs. Predicted Duration Comparison Across Models

Figure 8 compares actual versus predicted durations for *XGBoost*, Linear Regression, and PCA + Linear Regression. The dashed red line indicates an ideal fit where predicted values match actual durations.

The *XGBoost* model (in blue) aligns closely with the ideal line, demonstrating higher prediction accuracy. In contrast, predictions from simple linear regression (in green) and PCA + Linear Regression (in orange) exhibit greater scatter, particularly for longer durations, indicating their inability to effectively capture complex interactions in the data.

4 CONCLUSION

In this work, we addressed the challenge of optimizing distributed pipeline execution in serverless environments using an *XGBoost*-based approach. By predicting optimal configurations, our model successfully reduced execution time and costs on cloud platforms *Lithops*, achieving up to a 79.9% reduction in execution time and around 30% cost savings compared to an exhaustive *Design Space Analysis* (DSA). This significantly enhanced both efficiency and cost-effectiveness.

Serverless computing presents unique challenges for runtime estimation due to its dynamic and ephemeral nature. Unlike traditional environments, serverless platforms automatically scale

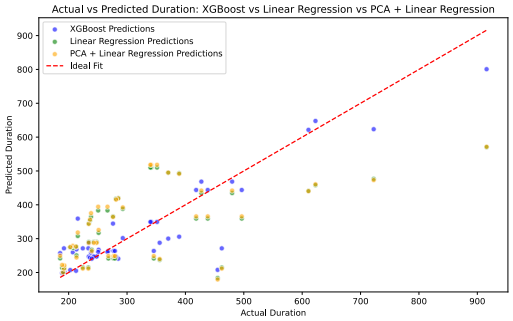

Figure 8: Comparison of actual vs. predicted duration for various models, highlighting the performance of *XGBoost* against simpler regression methods.

resources, but this scaling is not always linear or predictable. It depends on factors such as cold starts, variable latency, and cost-to-performance trade-offs across different configurations. Our approach addresses these challenges by effectively learning from limited configuration data to predict optimal settings.

Furthermore, the model leverages Lithops' ability to abstract backend differences, ensuring consistency in key parameters (e.g., memory, vCPUs, ephemeral storage) across FaaS platforms like AWS Lambda, Google Cloud Functions, and Azure Functions. This adaptability facilitates cross-provider deployment and effective optimization, independent of the underlying cloud provider.

Additionally, this approach recovers the initial training investment of 38.75 USD within two months, leading to substantial long-term savings.

4.1 Main Contributions

- Optimal Configuration Prediction: The *XGBoost* model predicts configurations that minimize execution time, achieving up to 30% cost reduction compared to *DSA*.
- Enhanced Performance Metrics: The model surpassed all baseline approaches, with a 75.34% improvement in *MAE* over *Baseline (Average)*, and 69.26% over *Linear Regression*, and 69.47% over *PCA + Linear Regression*, effectively capturing complex parameter interactions.
- Cost Efficiency: The model reduces execution times by up to 79.9% and costs by approximately 30%, requiring 30% fewer executions compared to exhaustive methods like *DSA*. For example, running 50 experiments instead of 100 results in a 50% cost reduction.
- Universal Applicability: The model's adaptability and standardized parameter collection via Lithops enable deployment and optimization across FaaS platforms, including AWS Lambda, Azure Functions, and Google Cloud Functions, ensuring consistent optimization for any pipeline configuration.

4.2 Future Directions

As the dataset grows, the model's accuracy and generalization can improve. While techniques like ensemble learning have been explored, further research into advanced architectures could boost predictive performance and refine optimization.

Additionally, the approach also has the potential to standardize the optimization of distributed pipelines across cloud platforms, offering a balance between accuracy, cost, and operational efficiency.

ACKNOWLEDGEMENTS

This work has been partially funded by the CLOUDLESS project, a platform for edge computing information, focusing on next-generation cloud and edge infrastructure and services, specifically the UNICO I+D CLOUD 2022 subproject. It is financed by the Ministry of Economic Affairs and Digital Transformation and the European Union - NextGenerationEU, within the framework of the PRTR and MRR.

Additionally, this work has been supported by the European Union through the Horizon Europe CloudSkin project (101092646).

We would also like to express our gratitude to the reviewers for their invaluable feedback, which greatly enhanced the quality of this work. Usama Benabdelkrim-Zakan is the corresponding author, along with co-authors Germán Telmo Eizaguirre-Suárez and Pedro García-López, both affiliated with Universitat Rovira i Virgili.

REFERENCES

[1] L. Cloud, "Lithops - serverless cloud computing framework," 2023. [Online]. Available: https://lithops-cloud.github.io/

[2] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," in *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 2017, pp. 445–451. [Online]. Available: https://pywren.io

[3] A. W. Services, "Aws lambda documentation," 2024. [Online]. Available: https://aws.amazon.com/lambda/

[4] C. Project, "Water consumption geospatial use case," 2023. [Online]. Available: https://github.com/cloudbutton/geospatial-usecase/tree/main/water-consumption

[5] Microsoft, "Azure functions documentation," 2024. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-functions/

[6] G. Cloud, "Google cloud functions documentation," 2024. [Online]. Available: https://cloud.google.com/functions

[7] A. Arjona, P. García-López, and D. Barcelona-Pons, "Dataplug: Unlocking extreme data analytics with on-the-fly dynamic partitioning of unstructured data," in *Proceedings of the 24th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID'24)*. IEEE/ACM, May 2024, oral presentation.

[8] H. Bannazadeh, M. Rabinovich, M. Chowdhury, and A. Wani, "Sizeless: Predicting the optimal size of serverless functions," in *Proceedings of the 21st International Middleware Conference*. ACM, 2020, pp. 56–70.

[9] C. Harris, K. Millman, S. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. Smith *et al.*, "Numpy: The fundamental package for array computing with python," *Nature*, vol. 585, pp. 357–362, 2020. [Online]. Available: https://numpy.org/

[10] T. P. D. Team, "Pandas documentation," 2024. [Online]. Available: https://pandas.pydata.org/

[11] Mapbox, "Rasterio documentation," 2024. [Online]. Available: https://rasterio.readthedocs.io/

[12] S. Community, "Shapely documentation," 2024. [Online]. Available: https://shapely.readthedocs.io/

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://scikit-learn.org/stable/

[14] O. Team, "Optuna: A hyperparameter optimization framework," 2024. [Online]. Available: https://optuna.org/

[15] M. Corporation, "Lightgbm: A fast, distributed, high performance gradient boosting framework," https://github.com/microsoft/LightGBM, 2017, accessed: 2024-01-01.