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

***Abstract*—Efficient execution of distributed pipelines in serverless environments is a key challenge to reduce both time and operational costs in the cloud. This paper presents an approach to predict and optimize the duration of a serverless pipeline, using a geospatial water consumption analysis pipeline as a case study, which is executed and parallelized with Lithops. 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.**

I. Introduction

Serverless data analytics pipelines are widely used for processing large datasets due to their scalability, cost-effectiveness, and ease of use. They enable developers to focus on writing code without managing infrastructure. Platforms like *Lithops* [1], an evolution of *Pywren* [2], execute Python functions as serverless tasks across multiple cloud providers, automatically handling resource provisioning and scaling. These platforms use rounds of *map* functions that interact with object storage, simplifying the development of data-intensive applications requiring dynamic scaling and parallelism.

However, optimizing the performance and cost-efficiency of serverless data analytics pipelines remains challenging. Execution time and costs depend on configuration parameters such as memory, vCPUs, ephemeral storage, and parallelism (e.g., splits). Finding the optimal configuration is complex due to the many possible parameter combinations. Exhaustive testing through *Design Space Analysis* (DSA) is costly and inefficient, as it involves multiple pipeline executions with varying settings.

A. Problem Statement

The challenge is to efficiently find the optimal configuration for serverless data analytics pipelines that minimizes time and costs, without testing all parameter combinations. 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.

B. Proposed Solution

We propose a machine learning-based approach to predict pipeline execution time based on key configuration parameters. By training a predictive model on a limited set of initial executions, we estimate the performance of unseen configurations. This enables us to efficiently identify optimal settings that minimize both execution time and cost, without exhaustive experimentation. Our model considers parameters such as the number of splits, memory, vCPUs, and ephemeral storage, and is designed to be universally applicable to any pipeline running on serverless platforms like *Lithops*.

To validate our approach, we use a real-world water consumption analysis pipeline for agriculture in the Murcia region of Spain [3]. This pipeline processes geospatial climate data to perform raster interpolations—a computationally intensive task that benefits from parallel execution. The pipeline consists of several stages:

1) Data Preparation: Uploading and converting Digital Terrain Models (DTMs) and geospatial data.
2) Raster Data Interpolation: Parallel interpolation of climate variables like temperature, humidity, wind speed, and solar radiation.
3) Computation of Potential Evaporation: Calculating evapotranspiration using interpolated data to estimate water consumption.
4) Result Visualization: Generating visual representations of the computed data.

C. State of the Art

Previous studies have explored resource optimization in serverless environments. Pimpley *et al.* [4] applied machine learning to optimize serverless queries, while Arjona *et al.* [5] developed *Dataplug* for dynamic partitioning. However, both approaches required exhaustive testing and had a limited scope.

Other works, such as [6], used models like *LightGBM* to estimate processing times in manufacturing but faced limitations due to small datasets, which we also encounter as a challenge, given the costs and difficulties in obtaining extensive data. Despite these limitations, our approach using *XGBoost* achieves better predictive accuracy and metrics.

Our model optimizes multiple parameters (e.g., memory, vCPUs, parallelism) simultaneously without exhaustive testing, resulting in significant time and cost savings.

D. 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 the DSA method. Our model optimizes multiple parameters simultaneously and is applicable to any serverless data analytics pipeline, offering a versatile, efficient solution.

II. Methodology

The methodology used to predict and optimize the duration of distributed pipelines in serverless environments includes several key phases: configuration analysis through *Design Space Analysis* (DSA), data collection and preprocessing, hyperparameter optimization, and training and evaluation of the predictive model.

A. Configuration Analysis Using DSA

The *Design Space Analysis* (DSA) involved executing the pipeline with 148 different configurations to understand how various parameters affect execution time. By systematically varying key parameters, we aimed to identify their impact on performance and resource utilization.

The parameters adjusted during the DSA include:

- Splits: Number of splits for parallel processing (2 to 6). More splits increase parallelism but may add overhead.
- Allocated Memory: Memory per function (1,024 MB to 3,008 MB). Higher memory improves efficiency but raises costs.
- Ephemeral Storage: Temporary storage (512 MB to 8,192 MB) for intermediate data.
- vCPUs: Virtual CPUs (0.85 to 1.61), affecting computation speed.
- Input Files and Size: Tested with 5 or 15 files, and sizes from 0.25 GB to 1 GB to assess data volume impact.

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

TABLE I
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

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.

B. 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 I) and additional features created through feature engineering (Table II), which capture complex relationships between the original parameters.

TABLE II
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: The data was split 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.

C. 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 III.

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 III
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

This combination of hyperparameter tuning and robust training ensured that *XGBoost* provided accurate predictions with a good balance between complexity and performance.

D. Predicting Optimal Configurations

With the predictive model trained on the data collected during the DSA (Sections II-A and II-B), we can now use it to optimize pipeline execution time based on specific configuration parameters. The model takes as input both the original parameters (Table I) and the derived features (Table II), as detailed in previous sections.

The terms *splits* and *chunk size* refer to dividing input data for parallel processing, determining the level of resource provisioning. More splits or smaller chunk sizes increase parallelism by distributing the workload across more resources, allowing for faster execution, though at the potential cost of higher coordination overhead. The model treats both terms equivalently in terms of optimizing resource allocation.

Our goal is to predict execution times for various configurations without having to run the pipeline each time, allowing us to efficiently find the optimal settings that minimize both execution time and cost.

For example, let's consider a pipeline that processes 24 input files, totaling 20 GB of data, and aims to optimize its execution on *AWS Lambda* using 1 vCPU. We set the input parameters accordingly, within the ranges observed during the DSA:

TABLE IV
EXAMPLE PIPELINE CONFIGURATION

Parameter	Value
`num_files`	24
`input_size_gb`	20
`runtime_memory_mb`	1,769 (for 1 vCPU)
`ephemeral_storage_mb`	512
`worker_processes`	1
`invoke_pool_threads`	64
`vcpus`	1

In this case, the key parameter of interest is the number of *splits* (also referred to as *chunk size* in other contexts), which determines the level of parallelism during execution. Although we use the term *splits*, the model treats both terms equivalently, as they both represent data division for parallelism.

We can use the model to predict execution times for different numbers of splits while keeping other parameters constant. For instance, testing splits ranging from 3 to 6 gives the following results:

TABLE V
PREDICTED EXECUTION TIMES FOR DIFFERENT NUMBERS OF SPLITS

Number of Splits	Predicted Execution Time (s)
3	200
4	180
5	170
6	190

As shown in Table V, using 5 splits yields the shortest execution time of 170 seconds, making it the optimal configuration for this parameter.

We can also adjust other parameters. For example, keeping the number of splits at 5, we varied the allocated runtime memory, as shown in Table VI. Allocating 2,048 MB results in the shortest execution time of 170 seconds, with no significant improvement beyond that, making it the optimal memory setting. However, in the case of using *AWS Lambda*, adjusting the allocated memory also directly impacts the number of vCPUs, as AWS Lambda dynamically allocates vCPUs in proportion to the memory assigned. Therefore, any changes in memory allocation will also affect the computational capacity through vCPUs, and this must be considered when optimizing the configuration.

TABLE VI
PREDICTED EXECUTION TIMES FOR DIFFERENT MEMORY ALLOCATIONS

Runtime Memory (MB)	Predicted Execution Time (s)
1,024	210
1,536	180
2,048	170
2,560	175

This process can be automated using a custom framework, similar to *Optuna*, allowing users to explore model predictions by defining parameter ranges and fixing others. Users could set ranges for configuration parameters, specify the number of trials, and automatically evaluate the predicted execution times. This would streamline experimentation, helping to identify optimal settings for minimizing execution time and cost, ultimately saving time and resources in optimizing pipeline configurations.

E. Model Improvement Attempts

Various strategies were tested to improve the predictive model, but none surpassed the final *XGBoost* approach. Key strategies included:

- CTGAN for Synthetic Data: Generated data didn't improve metrics, so the real data was used exclusively.
- Alternative Models: *Random Forest*, *LightGBM*, and *MLPRegressor* were evaluated. *XGBoost* outperformed all, with *LightGBM* showing the worst results.
- Ensemble Models: These showed an *MAE* of at least 50, worse than *XGBoost*.
- RFE for Feature Selection: This worsened performance, increasing the *MAE* to 51.59.
- Early Stopping: Implemented to prevent overfitting.

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 VII
RESULTS OBTAINED WITH DIFFERENT MODELS AND TECHNIQUES

III. RESULTS AND DISCUSSION

This section compares three main approaches for predicting and optimizing the duration of the geospatial analysis pipeline:

- *XGBoost* Model: A supervised learning model based on gradient boosting, trained with real data generated to predict the pipeline execution durations.
- *Baseline* Model: An approach that uses the average execution times from the training set for predictions. It lacks advanced predictive capabilities and is used as a reference.
- *Design Space Analysis* (DSA): An exhaustive exploration of the configuration space, manually testing different combinations of pipeline parameters and selecting the configuration that minimizes execution time.

A. Comparative Results Between Approaches

The tables below show the comparison of results obtained by the three approaches:

TABLE VIII
XGBoost MODEL METRICS.

Metric	Value
MAE on test set (s)	29.81
MAPE on test set (%)	8.72%
R^2 on test set	0.8802
Average MAE in cross-validation (s)	34.20

TABLE IX
Baseline MODEL METRICS.

Metric	Value
MAE on test set (s)	120.90
Improvement of *XGBoost* over *baseline* (%)	75.34%

TABLE X
MINIMUM DURATION SELECTED VIA *DSA*.

Metric	Value
Minimum selected duration (s)	184.08

Clarification: The *MAE on the test set* and *Average MAE in cross-validation* for *XGBoost* refer to performance on the test set and during cross-validation, respectively. The *baseline* uses the average duration, and the *DSA* selects the configuration with the shortest real duration based on the actual executions.

Analysis:

- *XGBoost* vs. *Baseline*: The *XGBoost* model showed a significant improvement of 75.34% over the *baseline* model in terms of MAE. This demonstrates *XGBoost's* ability to capture complex relationships between pipeline variables, providing more accurate predictions compared to the simplistic average-based approach of the baseline.
- *XGBoost* vs. *DSA*: Both *XGBoost* and *DSA* approaches identified configurations with short execution durations. However, *XGBoost* was trained with only 70% of the data used by DSA and could generalize well enough to predict the remaining 30% of unseen configurations (test set) with high accuracy. The *XGBoost* model achieved an MAE of 29.81 seconds and a MAPE (Mean Absolute Percentage Error) of 8.72%, demonstrating its ability to predict the most efficient configurations without running all combinations, unlike DSA.
- Cost Comparison: The *DSA* approach involves running all possible configurations, which, for the water consumption pipeline, resulted in a total cost of approximately 55.36 USD. In contrast, since the *XGBoost* model only requires 70% of the data to train and predict the optimal configurations, the cost for generating the training data is reduced by 30%, resulting in an estimated cost of around 38.75 USD. This showcases the cost-efficiency of the *XGBoost* model while maintaining a high prediction accuracy.

B. Comparison of Real vs. Predicted Duration

To assess the *XGBoost* model's ability to identify the optimal configuration, the model was run on all configurations generated through the *Design Space Analysis* (DSA), including both seen and unseen configurations during training.

The configuration with the shortest real duration was removed from the training set to test whether the model could correctly predict it as the most efficient configuration. This configuration was then added to the test set, where all durations were predicted. The result shows that the configuration with the shortest predicted duration corresponds to the same one with the shortest real duration.

Below is the table with the minimum pipeline configuration, along with the comparison between the real and predicted duration by the *XGBoost* model.

TABLE XI
COMPARISON OF THE CONFIGURATION WITH THE SHORTEST REAL AND PREDICTED DURATION BY *XGBoost*.

Parameter	Value
Splits	5
Number of Files	5
Total Input Size (GB)	0.25
Allocated Memory (MB)	2000
Ephemeral Storage (MB)	1024
Threads per Invocation	64
vCPUs	1.13
Real Duration (s)	184.08
Predicted Duration (s)	195.26

Analysis: The *XGBoost* model correctly identified the configuration with the shortest real duration, predicting a duration of 195.26 seconds compared to the real 184.08 seconds. Although the prediction is close, the difference highlights the need to continue refining the model to improve its accuracy. With more data, predictions may vary, but this would also offer

opportunities to fine-tune the model, enhancing its generalization and accuracy in future predictions.

Additional Results: When comparing two configurations with the same number of files and input size, as shown in Table XII, the model is capable of suggesting a configuration that minimizes the execution time by 79.9%, showcasing the potential for significant optimization.

Configuration with minimum duration:

TABLE XII
CONFIGURATION COMPARISON: MINIMUM VS. MAXIMUM DURATION

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
Worker Processes	1	1
Invoke Pool Threads	64	64
vCPUs	1.13	0.58
Memory per File (MB)	400	204.8
Storage per File (MB)	204.8	204.8
vCPUs per File	0.226	0.116
Files per vCPU	4.42	8.62
Size per File (GB)	0.05	0.05
Memory per GB (MB)	8000	4096
vCPUs per GB	4.52	2.32
Storage per GB (MB)	4096	4096
Threads per Worker	64	64
Memory per Thread (MB)	31.25	16
vCPUs per Thread	0.0177	0.0091
Memory per Thread vCPUs Ratio	1769.91	1765.52
Duration (s)	184.08	915.89

With the same input files, number of files, and similar parameters, the model suggests a configuration that reduces the execution time by 79.9%, thanks to a better distribution of resources, such as memory, number of splits, and vCPUs.

C. Learning Curve

Figure 1 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 small gap between training and test error demonstrates that the model is not overfitting and has good 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.

D. Distribution of Relative Errors

Figure 2 shows the distribution of relative errors in the test set. The model showed low errors in most predictions, with some scattered points corresponding to configurations with longer durations, which were difficult to predict accurately. However, these points represent a very small percentage of the total configurations.

E. Residual Analysis

The residual analysis, shown in Figure 3, reveals that while most residuals are symmetrically distributed around zero,

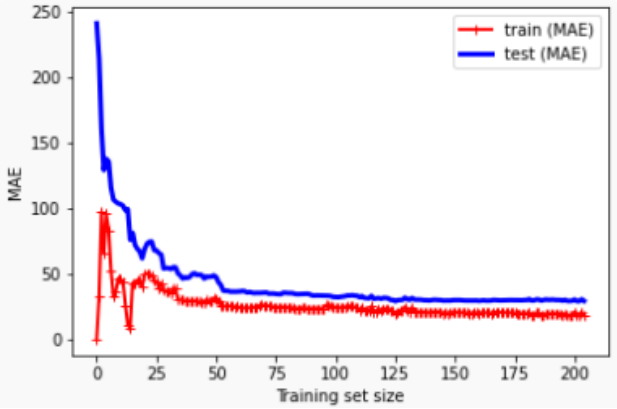

Fig. 1. Learning curve of the *XGBoost* model.

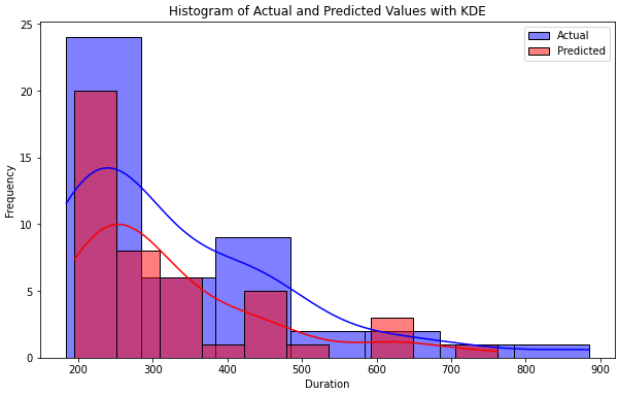

Fig. 2. Distribution of relative errors in the test set.

indicating no significant bias, there are a few points, not only from long durations, that deviate notably from zero. This further motivates us to continue improving the model.

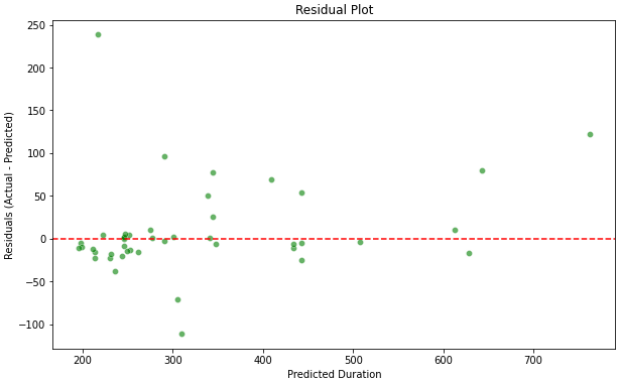

Fig. 3. Residual plot for the *XGBoost* model.

F. Real vs. Predicted Duration for Optimal Configurations

Figure 4 shows the comparison between the real and predicted duration for the optimal configurations. Most of the points are close to the identity line, indicating that the model correctly predicts configurations with shorter durations.

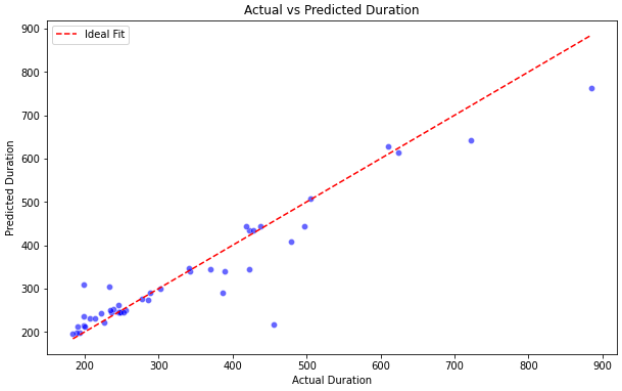

Fig. 4. Comparison of real vs. predicted duration for optimal configurations.

IV. Conclusion

In this work, we have addressed the challenge of optimizing the execution of distributed pipelines in serverless environments, using an *XGBoost*-based approach to predict and optimize the duration of these pipelines. By predicting optimal configurations and improving execution metrics, the study presents an efficient solution to reduce the time and operational costs associated with execution on cloud platforms like *Lithops*. Specifically, the model has demonstrated the ability to reduce execution time by up to 79.9% for the same tasks, and reduced overall costs by approximately 30% compared to the exhaustive DSA, significantly lowering both waiting times and operational costs.

A. Main Contributions

- Prediction of optimal configurations: The *XGBoost* model has proven effective in predicting optimal configurations that minimize pipeline execution time. By training the model with a subset of data generated through *Design Space Analysis* (DSA), it was possible to predict efficient configurations without running all combinations, reducing execution costs and computation time by up to 30%.
- Improvement in performance metrics: The model significantly outperformed the baseline, with a 75.34% improvement in *MAE*, demonstrating its capability to capture complex interactions between pipeline parameters such as memory, vCPUs, and the number of splits. These results underscore the model's potential to accurately predict unseen configurations.
- Cost savings in expensive pipelines: The model can reduce execution times by up to 79.9% and cut costs by approximately 30% compared to exhaustive methods like *Design Space Analysis* (DSA), requiring 30% fewer executions. For example, running 50 experiments instead of 100 would yield a 50% cost reduction. This is a substantial advantage for optimizing time and expenses in costly pipeline executions.
- Universal applicability and automation: The automated process of executing pipelines and collecting data makes this work applicable to any serverless data analytics pipeline. By enabling efficient analysis of resource usage, such as memory and vCPUs, this approach allows for precise calculation of task durations and optimal configuration identification, making it adaptable across various serverless platforms and use cases.

B. Future Directions

As more data becomes available, the model can continue to improve its accuracy and generalization capability. Although different techniques, such as ensemble learning, have been tested to enhance the model, further exploration of advanced architectures should be pursued to improve predictive performance and optimize pipeline configurations even further.

Additionally, the approach presented here has great potential to standardize the optimization of distributed pipelines in cloud serverless platforms, offering a balance between accuracy, costs, and operational efficiency.

References

[1] L. Cloud, "Lithops - serverless cloud computing framework," 2023, accessed: October 7, 2024. [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] C. Project, "Water consumption geospatial use case," 2023, accessed: October 2024. [Online]. Available: https://github.com/cloudbutton/geospatial-usecase/tree/main/water-consumption

[4] A. Pimpley, S. Li, A. Srivastava, V. Rohra, Y. Zhu, S. Srinivasan, A. Jindal, H. Patel, S. Qiao, and R. Sen, "Optimal resource allocation for serverless queries," in *Proceedings of the 24th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID'24)*. IEEE/ACM, May 2024. [Online]. Available: https://ar5iv.labs.arxiv.org/html/2107.08594

[5] 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.

[6] H. Yamashiro and H. Nonaka, "Estimation of processing time using machine learning and real factory data for optimization of parallel machine scheduling problem," *Procedia Computer Science*, vol. 221, pp. 227–234, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214716021000178