Improving Elastic FL Efficiency in Serverless Environments using Dynamic Resource Allocation and Intelligent Client Selection

MSc Research Project
MSc Cloud Computing

Silver Stalan Inbaraj
22105441

School of Computing
National College of Ireland

Supervisor: Sai Emani

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Silver Stalan Inbaraj

Student ID: 22105441

Programme: MSc Cloud Computing

Year: 2024 - 2025

Module: MSc Research Project

Supervisor: Sai Emani

Submission Due Date: 15/09/2025

Project Title: Improving Elastic FL Efficiency in Serverless Environments using Dynamic Resource Allocation and Intelligent Client Selection

Word Count: 9,569 **Page Count**: 22

Signature: Silver Stalan Inbaraj

Date: 15.09.2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only

Signature:	
Date:	
Penalty Applied (if applicable):	

Improving Elastic FL Efficiency in Serverless Environments using Dynamic Resource Allocation and Intelligent Client Selection

Silver Stalan Inbaraj
22105441

Abstract

Federated Learning (FL) has emerged as a pioneering form of distributed machine learning methodology that allows for the collaborative training of models over widely different devices with privacy of the data kept in mind. The traditional FL on top of public clouds is notably problematic and includes resource underutilization, the existence of the straggler effects, and poor client selection strategies especially in heterogynous settings that comprise computing capabilities and data distributions. The proposed research gives an original concept of the elastic FL framework that combines AWS autoscaling benefits with the statistical analysis algorithm to implement dynamic resources allocation scheme within serverless computing frameworks. The framework uses asynchronous aggregation strategy with a score-based client selection scheme that analyses the capacity and capability of the client, the nature of the data and the previous performance evaluation of the client. The Resource Optimizer uses Median Absolute Deviation (MAD) statistical analysis and policy-based scaling reasoning to optimize the use of resources, which does not require detailed machine learning predictions. Thorough comparison with the Vertical Pod Autoscaler in Kubernetes obtained a smaller training time (30%), saved cost (40%), and improved resource usage (21%) compared to the current best practices, making the case that statistical methods can be effective to optimize serverless FL.

1 Introduction

1.1 Research Background and Statement

The ever-growing number of edge devices and the Internet of Things (IoT), in general, has created unprecedented amounts of distributed data, requiring new solutions to machine learning that are sensitive to privacy restrictions but nonetheless take advantage of collective intelligence (Aminizadeh et al., 2023). FL is a revolutionary framework that allows joint model learning while each of its distributed clients optimizes its parameters; it supports applications in data privacy, motivated by a wide range of scenarios, including healthcare, finance, mobile computing, and many more. Nonetheless, these challenges are presented by the heterogeneity of the involved devices that introduce the system heterogeneity problem in regard to computational capacities, network connectivity, and data quality (Loconte et al., 2025). The straggler effect, which occurs in traditional synchronous FL, also makes these algorithms inefficient in the usage of resources and slow convergence. The recent increase in serverless computing technologies promises to overcome these difficulties with its event-based, auto-scaling functionalities that will be capable of dynamically scaling with different workloads (AlQassem et al., 2024). Combining FL with serverless architectures will be a paradigm shift to enhance more efficient, economically distributed machine learning protocols that can also auto-scale systems based on time-varying resource demands, even as privacy is formalized.

1.2 Motivation

The current state of FL systems is to a large extent associated with severe shortcomings of managing distributive resources and coordination between clients; in situations involving multiple devices of disparate computing capabilities, memory size, and network connection (Soltani et al., 2024). The straggler issue is a central problem and setting synchronous techniques causes the whole system to wait on the slowest participants causing massive waste of resources and suboptimal training efficiency (Alahyane et al., 2025). Conventional autoscaling systems tailored to web applications with unpredictable workloads are not optimal to apply to FL workloads with unpredictable tolerances of the resource needs in burst computing during model training (Tari et al., 2024). Also, classic client selection strategies become obtuse when applied to unstructured selection criteria due to the unrealistic heuristics or random sampling solutions they employ (Al-Saedi et al., 2025).

1.3 Research Question

What are the ways one could combine AWS autoscaling functionality with smart client selection algorithms and asynchronous aggregation protocols to improve the efficiency and scalability of FL systems in a serverless computing model relative to a traditional container-based implementation?

1.4 Problem Solution

This study is a proposal of a full elastic FL architecture which incorporates the three aspects of the identified challenges with three components connected together. To start with, a dynamic resource allocation system is based on the possibilities of AWS Lambda autoscaling proposing to combine custom scaling logic with the autoscaling capabilities of AWS Lambda containers to respond to concurrent workload nature and client demands through automatic computational resource allocation changes in the real-time environment (AlQassem et al., 2024). Second, an asynchronous aggregation engine can allow clients to provide updates to their model when they are ready to do so with having to wait and block the global training sufficient read on alleviating the straggler problem whilst providing the factors of convergence to models . Third, client selection algorithm selects participants on the basis of computational abilities, data quality indicators and past performance in order to maximize training efficiency and provide a fair participation among heterogeneous devices. The framework runs on serverless Function-as-a-Service (FaaS) platforms to enable event-driven execution models that automatically scale resources lower and higher depending on the clinical demands, and do not require manual capacity planning, allowing them to reduce operating costs of resources in use (Merlino et al., 2024). It is a resource-efficient form of FL that is infinitely scalable and can effectively manage different client populations and workload patterns by adapting to them and being able to scale at any one time.

1.5 Research Objective

The most significant research objective entails the development and quantification of a scalable elastic serverless FL architecture that determines resource allocations and customer involvement to maximize training performance, efficiency, and reduced cost with a relatively bigger impact on scalability across a range of heterogeneous scattered environments

1.6 Contributions

These are the research contributions this study offers:

- A new adaptive serverless FL architecture will be developed and deployed that will integrate AWS autoscaling features with built-in resource optimization efforts to deliver an adaptive workload management capability.
- The development of an asynchronous aggregation scheme in which delays due to stragglers are negated and guarantees of model convergence are achieved by using weighted averaging schemes
- The creation of a multi-criteria score-based algorithm of client selection which compares the capabilities of the hardware, data features and historical performance statistics to maximize training performance
- A distinctive evaluation framework where serverless approaches are compared to container-based solutions to a few FL scenarios with large-scale real-world data

2 Related Work

2.1 Serverless Computing and FL Integration

Elzohairy et al., (2022) solves the crucial straggler issue in serverless FL systems and forms a specific clustering-based semi-asynchronous training method (FedLesScan) that is designed to work on a FaaS platform. The study finds that conventional strategies of straggler mitigation cannot take into consideration distinct serverless features such as cold-starts, volatility in performance, and ephermeral stateless instances of functions. FedLesScan dynamically reacts with the behavior pattern of the clients, and enforces smart cluster technologies into to reduce straggler impact on the entire systems. Thorough tests based on Google Cloud Functions show that the training time is reduced by 8% and the cost by 20%, and with effective update ratios improving a substantial 17.75 percent over current methods.

Chadha et al., (2024) proposes an innovative asynchronous scoring-based approach to solving the problem of serverless FL and, in particular, with regard to the heterogeneity of hardware resources of the participating clients. The system has an advanced multi-dimensional scoring system, based on the capabilities of its clients hardware, the nature of their datasets, and their computational abilities to intelligently select clients to train during rounds. This method is a major improvement when compared with client selection strategies that are random or based on heuristics. The system has the advantage of reducing the straggler effects and allowing maximum utilization of the resources over various computing environments by balancing the resource loads during computing dynamically. The experimental findings show dramatic performance gains where the average speedup stands at 2.75x and the maximum speedup is 7.03x above the baseline methods. Also, the system eliminates cold start latencies by a significant margin and is thus very adaptable to serverless contexts where initialization latency of functions is paramount.

Qi et al., (2024) introduces a fully featured lightweight, event-driven serverless platform specially designed to accommodate a large-scale FL aggregation with the highest resource utilization. Using an elastic serverless architecture, the system resolves the innate resource inefficiency issue of the traditional always-on FL servers, which possess the fine-grained resource management capability. Locality-sensitive placement techniques used on the platform guarantee maximum use of shared memory benefits in conjunction with fine-grained scheme deployed in resource scaling. The architecture supports dynamic scalability basing on actual workload requirements instead of capacity that is previously provisioned. The experimental analysis has shown that both the resource usage and the speed of aggregation have been

increased substantially in comparison to practical serverful and traditional systems of serverless FL.
Chadha et al., (2024) deals with the severe shortcoming of typical global model architecture provisions in serverless FL systems, which employ heterogeneous client models with knowledge distillation approaches. The authors suggest an optimized serverless procedures of two Federated Learning via Model Distillation (FedMD) and Federated Learning Distillation for robust model Fusion (FedDF) prominent federated knowledge distillation methods by building on the FedLess structure. The research addresses the problems of resource and statistical heterogeneity with the selective architecture of the models. Extensive tests support the idea that serverless FedDF is more robust to highly skewed data distributions than FedMD.
Hu et al., (2023) introduces a multi-faceted serverless FL service ecosystem that can solve the problem of data and knowledge isolation on different cloud service providers in the collaborative multi-cloud environment. The system gives a capability to do customized model training and fine-grained sharing of models and preserves privacy in a multi-private cloud scenario. The architecture uses a complete lifecycle FL service framework with the flexibility to operate across a range of multi-cloud implementations and support model sharing that will get the best value out of trained models. The serverless computing principles are parts of the solution since the implementation does not come with constraints on the overheads of maintaining infrastructure and provides scalable and on-demand FL services. The ecosystem will also allow smooth interoperation between organizations, which have chosen different cloud providers, without the inconvenience of privacy or data security issues.

2.2 Dynamic Resource Allocation with Asynchronous FL

Mughal et al., (2024) considers an efficient method of selecting the edge nodes and allocating them resources efficiently in heterogeneous edge computing environment and proposes a new architecture called Multi-Edge Clustered and Edge AI Heterogeneous FL (MEC-AI HetFL). The proposed methodology will use schematizing improved multi-edge cluster algorithms and AI-empowered node connectivity to assist dynamic selection of computing nodes of significance and minimize complexity employed global learning mission optimization. The framework deploys asynchronous edge devices with synchronous Edge AI incremental mechanisms that allow intelligent selection of nodes based on extensive quality rating on the network, computation capabilities, and the level of AI expertise. The three-layered architecture has joint device placement and resource allocation algorithms taking both the hardware constraints and data characteristics into consideration. Compared to the existing methods, the experimental results show improvement with a three- to fivefold advantage in resource allocation efficiency and learning accuracy and outstanding 98.6 percent accuracy in reality.
Forootani and Mohammadi (2024) present a general solution known as the Asynchronous FL algorithm, which is suggested as a solution to the scalability issue and efficiency concerns associated with conventional synchronous FL solutions. The way is that clients can swap the global model without coordination and without waiting on slower clients, which is important when clients have varying capabilities and environments change frequently. The framework has stringent convergence analysis that considers the delay caused by the client and model staleness impacts that involve the learned martingale difference sequence theory, and variance bounds to have resilient convergence under asynchronous updates. Significant contributions are theories with convergence guarantees in the face of arbitrary delays of clients, and viable implementation strategies to keep in mind in systems to deploy in the real world. The research solves the underlying scalability issues regarding FL through the capacity to adjust its model continuously without being blocked by the synchronization problems, which is especially outstanding to large-scale distributed systems that have diverse client availability and computing potential.

Chu et al., (2024) proposed a dynamic resource scheduling algorithm that was specially designed to be used in asynchronous federated computing when dealing with lightweight digital twin-empowered IoT networks. Their solution deals with the multi-objective optimization problem of minimizing energy consumption and minimize latency under the constraints that the performance of the FL model has to be respected. The process is decomposed by the Lyapunov technique to make the complex optimization problem solvable into one-slot optimization problems. They obtain closed-form solutions to optimal transmit power on the IoT device side and a multi-armed bandit (MAB) framework to optimize a client utility-based upper confidence bound (CU-UCB) algorithm on the server side to deal with a lack of knowledge about the state. Numerical results indicate the superiority of benchmark schemes, faster training on Fashion-MNIST and CIFAR-10 datasets.
Wu et al., (2024) suggest an accurate communication approach to the asynchronous FL that responds to the shift of communication link conditions by adjusting training and uploading activity. Their method solves the heterogeneity encounters in FL where gadgets can expand training functions under unfavorable interconnection circumstances and publish models at more advantageous connections. The most significant novelty is a computable approach to aggregate weights with regard to model distances and amount of local optimizations to balance the errors caused by heterogeneous updates to models and maximize learning velocity. This method analytically shows a convergence, with an improved performance with more opportunities of optimization at the time of good link conditions that yield an absolute of 12 percent optimization increase, 5 percent communication resource decrease, and accuracy of learning improvement by 3 percent in the case of CIFAR10 tasks.
Qureshi et al., (2024) present asynchronous federated architecture (AF3N) of IoT-enabled UAV networks as a solution to the special needs of the unmanned aerial vehicles serving as the flying base stations. Their model allows the training of models locally and then pushing the parameters to the mobile-edge computing servers with the help of device selection strategies that improve the learning rate. Its methodology incorporates a shared resources allocation algorithm whose basis is multi-agent asynchronous advantage actor-critic (A3C) whose key aim is to minimize latency and energy level. The methodology augments the resource allocation task into the step problem of Markov decision process and shows better performance with human-competitive accuracy (over 97%) and does not excessively affect the system cost or energy usage levels, resulting in a 5.52 percent higher system reward.

2.3 Client Selection and Straggler Mitigation

In (Lang et al., 2024) the authors comment on the problem of Straggler in synchronous FL and implement SALF (Straggler-Aware Layer-wise FL). This approach uses the backpropagation process to permit the use of layer-wise updates to models, with the stragglers contributing the partial gradients of completed layers as opposed to dropping off. Its most important contribution is that it allows synchronous presence of stragglers, due to partial updates of gradient without compromising convergence guarantees. The results demonstrate that SALF can reach a similar accuracy with the vanilla FL that has no latency requirements but improve considerably over drop-stragglers approaches, especially when 90 percent of the users turn into stragglers. The method can also be proved to converge with the same speed as vanilla SGD, and in general represents a novel manner of retaining the convergence properties of synchronous training without preferentially losing slow clients.
Hard et al., (2024) examines how current implementation of FL algorithms deal with very delayed clients (minutes to days) via Monte Carlo simulations with real world applications. The methodology tests FedAvg, FedAdam and FedBuff algorithms to establish their weaknesses when dealing with straggler clients. Two new algorithms (FARe-DUST (Federated Averaging with Regularization via Distillation on Stale Teacher), based on knowledge

distillation and FeAST-on-MSG (Federated Asynchronous Straggler Training on Mismatched and Stale Gradients), based on exponential moving averages) make the most significant contributions. Compared to the existing methods, the results on EMNIST, CIFAR-100 and Stack Overflow datasets show improved straggler accuracy and improved accuracy-tiling trade-off on the one hand as well as on the other hand. The study offers practical solutions to practical applications of the FL deployment where availability of clients is unpredictable.

Study	**Strategy Type**	**Primary Objective**	**Technologies**	**Key Metrics**	**Implementation**	**Dataset/Workload**
Chadha et al. (2024)	Asynchronous Scoring-Based	Straggler mitigation, client selection	Google Cloud Functions	Speedup (2.75x-7.03x), training time reduction (8%)	Authentic deployment – Evaluated on Google Cloud Functions with authentic federated scenarios.	Various federated scenarios
Qi et al. (2024)	Event-Driven Dynamic	Resource utilization optimization	Elastic serverless platform	Resource usage, aggregation speed	Authentic deployment – Evaluated on an elastic serverless platform with large scale FL scenarios (AWS Lambda–like environment).	Large-scale FL scenarios
Chu et al. (2024)	Multi-objective optimization	Energy-latency trade-off	Digital twin-empowered IoT, MAB framework	Energy consumption, latency	Hybrid – Digital twin simulations with IoT testbed experiments.	Fashion-MNIST, CIFAR-10
Wu et al. (2024)	Communication-Aware Asynchronous	Communication efficiency	Adaptive training/uploading	Accuracy improvement (3%), resource reduction (5%)	Authentic deployment - Evaluated on a distributed CIFAR-10 federated learning implementation with adaptable communication.	CIFAR-10
Our Framework	**Statistical Analysis + Policy-Based**	**Multi-dimensional optimization**	**AWS Lambda, MAD statistics**	**Training time (-30%), cost (-40%), resource utilization (+25%)**	Authentic AWS cloud deployment - used AWS Lambda, DynamoDB, S3, and implemented using 40 simulated client profiles.	**CIFAR-10 with 40 client profiles**

Guo et al., (2024) resolves the straggler issue in using distributed coreset selection where each client forms representative subsets of data without joint cooperation. The privacy-preserving distributed coreset generation is possible with the adoption of the methodology which reduces coreset optimization to k-medoids clustering problems. The most significant one is FedCore algorithm that adaptively chooses coresets on a training round basis on perfect sample gradients of gradient descent. It shows 8x improvement in FL training time with no decrease in model accuracy relative to FedAvg and better performance in comparison to FedProx that uses diminished local epochs to control a straggler. The theory analysis is that convergence assurances may include coreset errors in gradient approximations. This solution is a paradigm shift of the time-based attempts of straggler mitigation toward the data-driven straggler mitigation.
Marfo et al., (2025) proposes the concept of an adaptive client selection scheme that is a mixture of both differential privacy and fault tolerance to assist in FL applications. The methodology dynamically adjusts the participation of the clients according to, on the one hand, the performance of a model and the constraints of a system, and, on the other hand, the privacy of users due to a noise addition. The major contribution is that it presented a comprehensive framework which has focused on both client selection properties and a privacy preserving scheme in fault tolerance. The experiments on UNSW-NB15 and ROAD datasets regarding the network anomaly detection show that the improvement in accuracy is 7 percent and training time is reduced by 25 percent with respect to the baseline methods. The methodology takes into account the utility-based scoring of clients based on the quality of data and the processing power. This framework is notably superior when it comes to cybersecurity uses of cases where reliability to clients and the sensitivity of the data are of utmost consideration.
The approach by (Jiang et al., 2023) proposes a FL within the heterogeneity-aware framework in regard to adaptive client sampling and gradient compression techniques. The methodology will take account of statistical heterogeneity by way of picking representative sets of client populations and allocating varying proportions of compression ratios addressed to heterogeneous client potentials. The major advantage is concurrent optimization of the variety of clients and efficiency in communication with the help of adaptive features. The method brings down the cost of communication and minimizes straggler effects since it lets the clients upload sparse model updates that align with their computing capabilities. The convergence efficiency is better and the overhead of communication is also lower than that in traditional federate learning-based methods.

Table-I: Comparison of Key Related Works

2.4 Research Gap Analysis

Through the in-depth literature review, there are apparent research gaps in terms of how current studies on serverless FL research limit the actual implementation and scaling of this kind of system. To begin with, approaches in use do not integrate cloud autoscaling. Unlike (Chadha et al., 2024) and (Qi et al., 2024) around the most effective client selection and lightweight serverless architecture restorations, respectively, neither of them offers AWS-native autoscaling, which allows effectively utilizing dynamic resources. Existing approaches are based on either static resource provisioning or primitive container orchestration and lack the potential of smart elastic scaling based on FL workload intelligence. Second, there is no full cloistering of mitigation stragglers. Even though FedLesScan (Elzohairy et al., 2022) considers stragglers using clustering, and (Wu et al., 2024) propose asynchronous aggregation, no current work considers multi-dimensional client scoring, asynchronous aggregation, predictive resource scaling, and their joint application. Most methods manage the stragglers in a reactive manner as opposed to intelligently allocating resources to avoid degradation of performance. Third, FL can run at variable workload, and this can be encapsulated at a lowest-cost.

Finally, the gap between the excellence of the better client selection algorithms and the elastic serverless infrastructure does not exist. Though each individual element seems promising, there is yet to be presented a comprehensive framework illustrating how federation through score-based client selection, asynchronous aggregation, and AWS autoscaling can integrate together synergistically to provide the best performance of FL in a production setting.

3 Research Methodology

3.1 Research Design

The proposed research will employ a holistic approach to the research problem in the assessment of the efficiency of adopting AWS autoscaling functionality and complementing it with bespoke scaling logic to flexibly allocate resources in the architecture of serverless FL. The methodology is a mixed-methods research study as it involves both the quantitative analysis of the system performance and the prospective systematic analyses of the architectures in order to integrate the intricacies encapsulating the elastic FL systems. Its research design includes the development of theoretical framework and its practical implementation validation using a coordination system of special elements such as the FL Coordinator, Client Manager, Resource Optimizer, and Model Aggregator, which are aimed at covering each specific layer of serverless FL optimization.

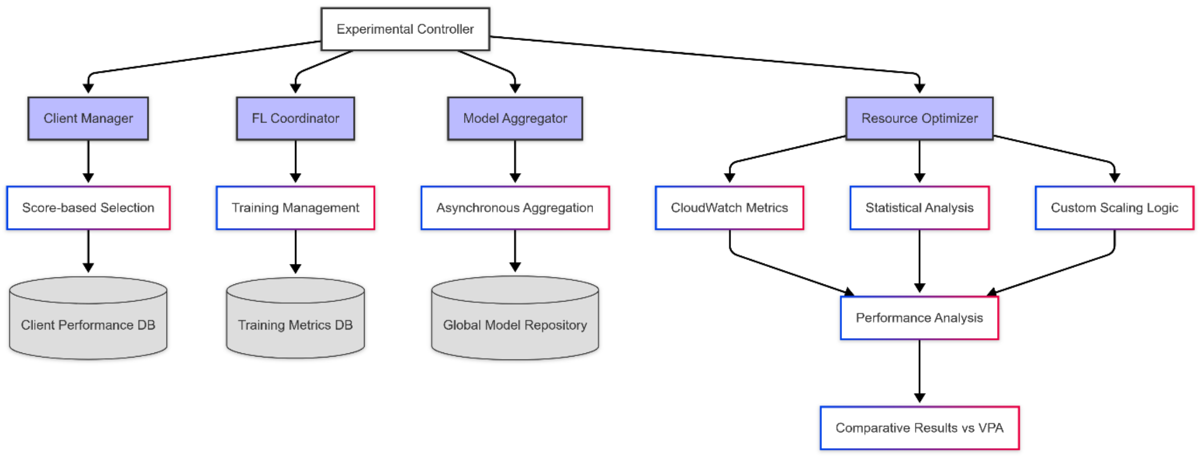


Figure 1: Proposed Research Methodology

3.2 Theoretical Foundations

The theoretical basis of this work draws on three major algorithmic building blocks that will facilitate successful serverless FL. FedAvg is the most commonly studied distributed machine learning aggregate algorithm. The typical FedAvg algorithm calculates the global model as a weighted average of the local models of the clients: $w^{(t+1)} = \Sigma(ni/n) \times wi^{(t+1)}$, where $wi^{(t+1)}$ is the global model weights at the client i after leading local training, ni is the amount of the training samples at the client i, and n means the overall amount of samples over all partner clients. The strategy makes clients with more extensive data bases have a proportionally larger impact on the overall global model update.

With heterogeneous serverless settings, the FedAvg protocols need to be augmented to cope with asynchronous updates, and unequal systems. The augmented formula includes staleness awareness weighting: $w^{(t+1)} = \Sigma[(ni/n) \times \alpha i \times \beta i] \times wi^{(t+1)}$, where αi is the staleness penalty factor and βi denotes the client reliability factor. This improvement means that the older updates are given less influence, but convergence is still guaranteed by the exponential code.

Straggler detection mechanism utilizes MAD as a resistant statistic detection of clients whose performance is abnormal. MAD is determined by: MAD = median(|Xi - median(X)|), where Xi refers to client completion times and X refers to the set of all the client completion times. MAD has advantages over standard deviation in close to equal outlier detection performance, as it is not impacted by outliers and thus appropriate in all heterogeneous serverless settings.

3.3 Methodology

The study uses an experimental design model with four main elements that interact with each other and provide elastic FL abilities. The FL Coordinator is the orchestration hub orchestrating training rounds and providing management of the entire FL lifecycle with smart job and training monitoring. The Client Manager component supports highly sophisticated score-based algorithms to select the clients based on their hardware capabilities as well as evaluation of the service on data characteristics, and historical performance metrics stored in the Client Metadata Database. The Resource Optimizer is the most fundamental realization of the study, combining the AWS CloudWatch metrics analysis with the custom logic of scaling and statistical analysis algorithms. The optimization algorithms employ mathematical expressions and multi criteria decision matrices to determine the best possible Lambda configurations.
The straggler detection algorithm utilizes robust statistics of MAD in determining the clients whose completion performances surpass statistical bounds of performance. The method can withstand outliers and completes the exact straggler classification decision for resource optimization decisions. The Model Aggregator will execute the asynchronous aggregation engine that mitigates the core issue of the straggler in stochastic environments of diverse FL. The asynchronous solution allows to constantly advance the model without having to wait on slower clients, which makes training much more efficient (and still guarantees convergence due to more algorithmic alternatives).

3.4 Data Preparation

The experimental design follows thorough data preparation guidelines that will be used to facilitate the assessment of the elastic FL framework in different scenarios, and with various datasets. The most comprehensive in terms of their coverage of evaluations are CIFAR-10 (used to perform standardized benchmarking of image classification tasks) and FEMNIST (that has a naturally stratified data of handwritten character recognition and contains inherent non-IID attributes). The different datasets are associated with different FL tasks, which allows full evaluation of the systems performance across different application domains and data distributions patterns.
The Client Metadata Database keeps elaborate profiles on simulated clients that tracks the processing speed, memory limitations and network reliability characterizes, and past performance levels. This detailed modeling of clients allows evaluating the score-based selection algorithm in a close-to-realistic scenario and how the system would behave under a wide range of participant characteristics.

3.5 Services Integration

The implementation approach uses inherent AWS services to establish a production-ready serverless FL platform that reflects a practical applicability of its use outside an academic research setting. AWS Lambda functions give the basic computational infrastructure of all of the components and configure the amount of memory per system and the execution waiting timeframe that may be fluidly reprovisioned according to the workload demands. Serverless architecture does not pose a drain of infrastructure maintenance but entails an autoscaling ability that scales-up and scales-down in response to changing FL demands.

AWS CloudWatch integration allows fully monitoring and gathering metrics that can be looked into by the Resource Optimizer during decision-making. The AWS Auto scaling services combine with the custom scaling logic to offer fluid rules of adjusting the resources that consider the nature of the workloads of FL. The high-performance storage architecture is Amazon S3 with which Global Model Repository is managed, and Amazon DynamoDB with which client metadata and train metrics are kept. S3 is used for the reliable scalable storage of model artifacts with an automatic versioning system allowing model version rollback and historical analysis needs. DynamoDB allows access to low-latency profile information about clients and measures of their performance to facilitate real-time selection decisions in the score-based algorithm and performance tracking at a client level to monitor performance across system components.

3.6 Evaluation Framework

The evaluation methodology includes the comprehensive evaluation of the performances in terms of many aspects such as the effectiveness of the models, the efficiency of the systems, the economic feasibility, etc. The evaluation framework monitors accuracy over training rounds and measures the global performance of the model as well as the client-specific performance of the model to discern the ability to personalize and federation advantages. The monitoring tool can measure patterns of CPU, memory, and network bandwidth usage of every component of the system, and guide efficiency on the use of resources and how to optimize them. The response time analysis is comprised of columnar breakdown of communication latencies, aggregation processing times, and resource allocation decision latencies to check on the bottlenecks and verify scaling efficiency. The cost analysis feature allows thorough economic analysis of the serverless FL deployment in terms of calculated cost of Lambda execution and usage of S3 and DynamoDB resources, as well as cost-per-accuracy-unit that would normalize costs against reached model quality.

4 Design Specifications

This chapter indicates the entire design specification of the elastic FL template deployed in serverless clouds. The architecture resolves the high-level issues of resource heterogeneity, straggler resilience, and dynamic scaling when it comes to distributed systems. The offered architecture uses AWS serverless features in order to build a cost-effective and scalable solution that ensures a high level of the proposed model accuracy and optimal use of resources.

4.1 System Architecture

The elastic FL design in Figure 2 follows a layered structure whereby responsibilities are anachronized, but all parts remain connected. There are five main layers to the architecture, which are the coordination layer, client management layer, aggregation layer, resource optimization layer, and monitoring layer. The different layers are independent of each other, even though they communicate using well-established interfaces so that modules can be replaced and that improvements may be possible.

The coordination layer executes the whole process of FL controlling the submissions of jobs, training rounds, and convergence tracking. This layer keeps the global state of all active training jobs, and it orchestrates communication between the participating clients and the aggregation service. At the client management layer, the client registration, the capability evaluation, and smart selection of training rounds takes place. This layer provides the ability to do the best possible selection of clients based upon training efficiency versus use of resources by keeping well-detailed profiles of both client capabilities and performance in the past.

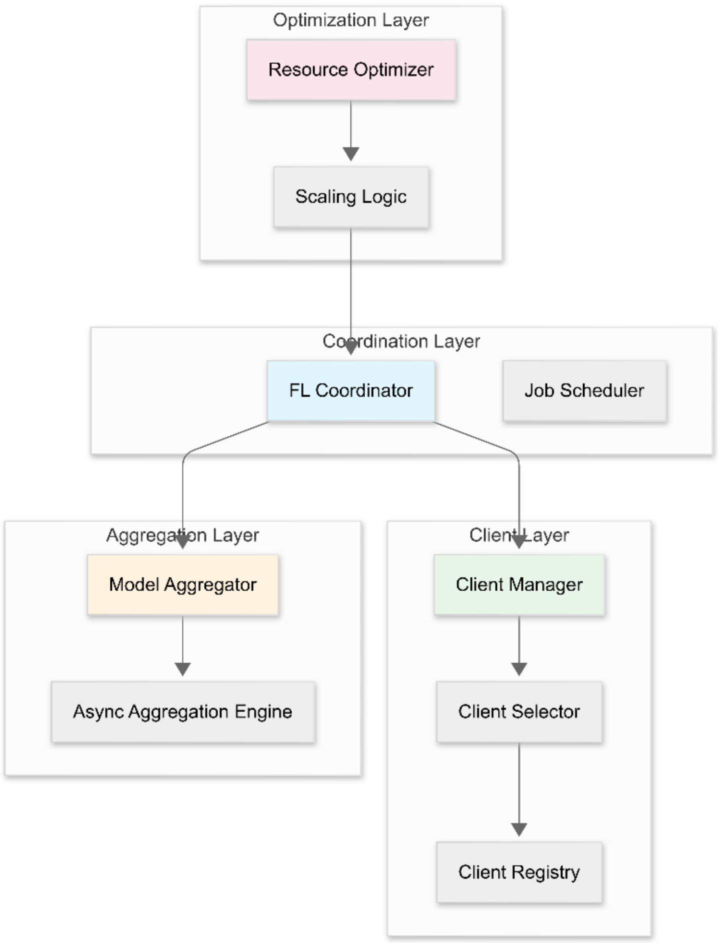


Figure 2: Proposed Serverless Elastic FL Architecture

4.2 FL Coordinator Design

The FL Coordinator is the central orchestration unit that oversees this lifecycle of FL jobs, starting with job submissions and continuing on till the job is complete. The coordinator performs a state machine that monitors the progression of jobs through phases that include initialization, selection of clients, training, aggregation and convergence evaluation. This structure guarantees secure job performance as well as being flexible on the various training approaches.

The coordinator stores job configurations in DynamoDB, so that a persistent state can be managed between Lambda invocations. The architecture of the model, training parameters, chosen clients, convergence criteria and track record of performance are always saved per job record. The design is robust in terms of fault tolerance due to state recovery and making detailed monitoring and analysis of the job possible. The job submission interface expects all the configurable parameters such as the model type, dataset specification, maximum training rounds, target accuracy, and client needs. The coordinator does the validation of such parameters with the systems capabilities and resource constraints before the job is accepted. The coordinator initiates the global model and kicks off the selection of clients on acceptance.

4.3 Client Management

The client management system adopts an advanced scoring system to rate clients through several dimensions including computational strength, network width, data nature and past history. This multi-criteria selection methodology creates a balanced client selection, which is efficient in terms of training effectiveness and model quality.

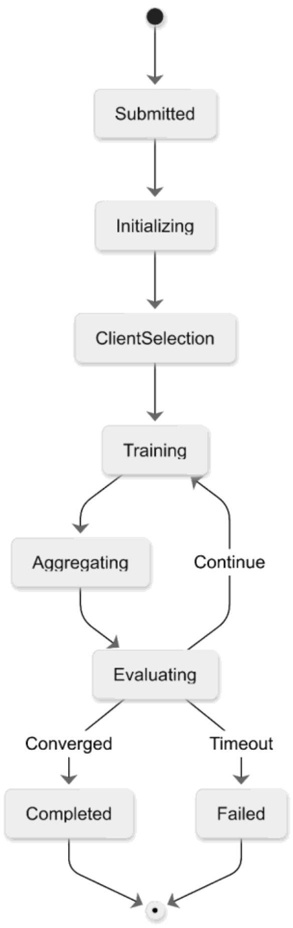


Figure 3: FL Coordinator Model

The algorithm of capability scoring puts different hardware characteristics under weights, and CPU cores and memory size as they directly affect performance in training are given higher weights. The bandwidth available to the network has a bearing on the score by considering the anticipated model size and the frequency of update. The scoring formula incorporates:

Score=0.3×CPU(norm)+0.3×Memory(norm)+0.2×Bandwidth(norm)+0.2×Performance(hist)

The client scoring parameters are defined as follows- CPU(norm) is the normalized count of CPU cores, Memory(norm) is the normalized memory capacity calculated as client memory/maximum memory capacity (32 GB), Bandwidth(norm) is normalized network bandwidth, Performance(hist) is historical measure of performance derived using past training completion time normalized to 0-1 scale with higher values triggering better historical performance.

This client selection method is a hybrid method in that it involves deterministic selection of high performing clients and probabilistic selection of client diversity. Such a method would certify in centrality of training but guarantee of model morphosis due to presentation with diverse distributions of data. To encourage fairness and to avoid client starvation, the selection algorithm maintains 60 percent of slots to use by clients that score the most and fills the other 40 percent by using weighted selecting.

4.4 Asynchronous Aggregation Engine

Asynchronous aggregation engine deals with straggler issue existing in heterogeneous FL settings. The asynchronous engine does not stop model updating and training waiting until all of the chosen related clients arrive as opposed to synchronous approaches that do. Updates and training are continuously corrected, and offline clients are excluded.
Since the aggregation strategy employs staleness-aware weighting then weights to updates that were long ago are diminished in relation to the global model. The weighting of updates is biased according to their temporal difference with respect to the latest version of the global model, so that old gradients do not cause training to become unstable. The penalty for the staleness is an exponential decays:

$$w_i = w_{\text{base}} \times e^{-\lambda \times \text{ staleness }_i}$$

The staleness penalty parameters are determined as follows: w_i is the adjusted weight on updating client i model, w_{base} is the base weight calculated in accordance with FedAvg proportionate weighting scheme based on client training data size, λ is the staleness decay coefficient empirically selected in the form of 0.1 as it strikes a satisfactory balance between model update freshness and participation equity, and staleness $_i$ is a measure of time, measured in training rounds, separating the client update time and the current global model update.
The engine keeps a slack window of model updates, allowing it to flexibly set aggregation policies that are a trade-off between model quality and training speed. The window size also grows or shrinks according to rates of client participation and the state of the network, growing when a lot of data is being exchanged and shrinking when there are less updates to be pushed

4.5 Resource Optimization Framework

The resource optimization framework uses the multi-level dynamic resource allocation process that functions at both job level and client level. The framework has an operational mode of continually evaluating resource consumption, performance measures, cost components in scaling the appropriate decisions that are optimal in terms of trade-offs between efficiency and performance. On the job level, the optimizer considers aggregate counters such as mean training time, straggler ratios and convergence rates to compute total resource requirements. The workload properties, including model complexity, the size of input data, and a batch, are considered by the optimization algorithm with the purpose of finding optimal Lambda settings:

Optimal Memory = Base Memory + Data Memory + Model Memory + Buffer

Optimal Timeout = Base Time × Complexity Factor × Safety Margin

The resource optimization parameters are as follows: Base Memory describes the basic Lambda execution environment footprint that is set to 128 MB, Data Memory is simply twice the input data size in MB to handle processing overhead and temporary storage needs, Model Memory is set to varying levels of complexity, where low, medium and high complexities require 128 MB, 256 MB and 512 MB respectively based on typical CNN memory requirements, and Buffer is used to provide extra headroom typically set between 10-20 percent of calculated requirements. The client-level optimization looks at individual performance variations, and finds clients that are outliers in poor performance, and suggests that there be some resources specifically altered with respect to them. The optimizer has maintained performance profiles that monitor the resources patterns to predict scaling to anticipate resource requirements before bottlenecks have already happened.

4.6 Scaling Logic and Policies

The scaling logic performs a policy-based mechanism, supporting workload-specific and goal-specific customizable scaling actions. It has several scaling policies such as performance-optimized, cost-optimized and balanced policies that each have different thresholds and actions

supported. The scaling policies establish limits on some of the important metrics such as resource utilization, training time, and the number of stragglers. The scaling logic reacts by making necessary decisions when thresholds set on metrics are crossed over a period of time. The policies provide cooldowns to avoid oscillation and make the system stable.
The manual scaling implementation supports the scaling recommendation generated by AWS Academy limitations allowing scaling recommendations to be applied by the administrators using Lambda console. These recommendations are captured in the system with comprehensive rationale and anticipated impact analysis in DynamoDB in order to assist in decision making.

5 Implementation

This chapter specifies how the elastic FL framework has been implemented with AWS serverless services as demonstrated in Figure. The implementation converts the design requirements into functional modules that are executed on the cloud platform of AWS by invoking Lambda functions as compute nodes, DynamoDB as the state management component, S3 as the model storage service and CloudWatch to provide monitoring and observability. It is implemented in a modular manner and consists of five major Lambda functions that all together will deliver the full FL capability. Each of the functions runs independently and exchanges information using asynchronous messaging and shared storage in order to be scalable and fault tolerant.

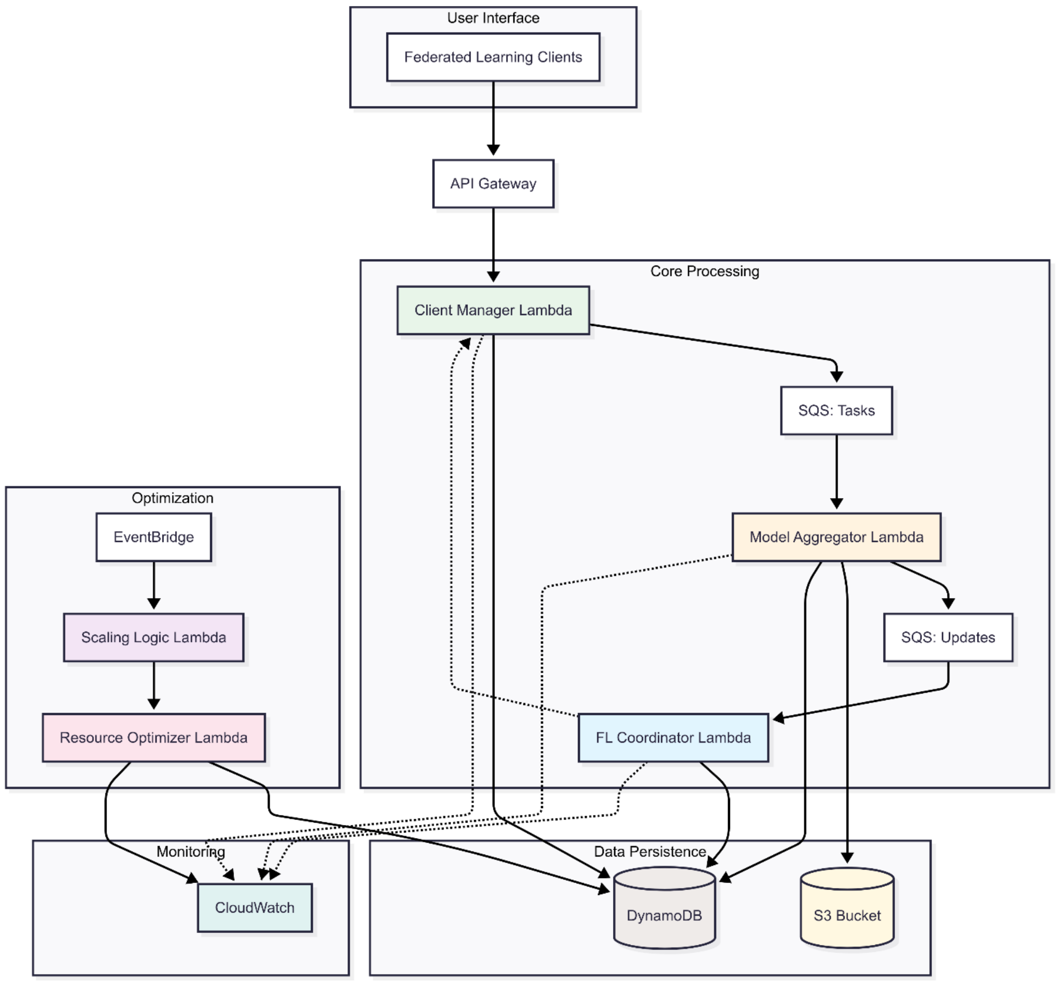


Figure 4: FL Implementation Architecture

5.1 Development Environment

The development platform was based on the use of Python 3.9+ as the main development language that provides a large machine-learning ecosystem and out-of-the-box compatibility with the AWS Lambda environment. The development process utilized Visual Studio Code as a text editor, Git as a version control management tool, as well as the AWS cli to deploy the code to the server automatically. The use of boto3 was implemented to integrate its use of AWS services so that Lambda, DynamoDB, S3, and CloudWatch could be accessed programmatically. The deployment methodology leveraged principles of infrastructure-as-code and all AWS resources were implemented and configured using Python scripts. This was to make the environment easily set up where reproducible deployments could be made and easy evaluation and testing of deployments.

5.2 FL Coordinator

The Lambda FL Coordinator service is the central orchestration point of FL jobs. This execution employs a state machine pattern to handle job lifecycle and job state(s) persisted on DynamoDB to make sure that the job state is accessible to new Lambda invocations. The coordinator processes five major functions, namely: submission of jobs, retrieving of status, initiation of training rounds, completion of round as well as setting up of datasets. The job submission handler validates job input parameters, generates a unique job and initializes the global model in S3.

The training round management uses checkpointing to allow recovering failures. The state of each round is saved prior to the selection of clients and following aggregation, so as to enable the system to recover to the last successful checkpoint. Training completion through the implementation is established through convergence metrics, which include improve accuracy, minimized loss, and parameter stability.

5.3 Client Manager

The Model Aggregator runs both the message processing implementation with SQS, and the algorithm called weighted averaging. The processes of implementation modelling update an SQS queue based on the arrival of a message and not on synchronous waiting. This strategy will allow aggregation on an ongoing basis with different times taken to complete by the clients.

The aggregation algorithm uses FedAvg that has been augmented to work in heterogeneous settings. The updates of the models are then weighted using the quantity of samples used in training and a staleness factor that limits the effect of the delayed updates. The application uses a sliding window of the last updates and can therefore use flexible aggregation policies balancing the model quality and training speed. It involves convergence detection that observes the accuracy increase and the decrease in the number of losses with the increasing rounds. The aggregator declares job completion after the convergence criteria are met, or maximum rounds are reached and the final model is saved to state in S3.

5.4 Resource Optimizer

The implementation of the Resource Optimizer gives detailed performance examination and resource suggestions. The optimizer assesses various dimensions of the performance such as training time, resource utilization, and straggler event over various resource configurations to find optimal resource settings. The implementation estimates the resources requirements relative to the characteristics of workload and makes recommendations relative to the observed performance.

Straggler detection algorithm detects clients whose completion times are beyond the statistical limit as compared to the median. The implication employs strong statistics that cannot be easily influenced by outliers since MAD is computed to determine the normal limits of performance. Clients who invariably stay beyond these limits are identified as stragglers and are pegged to be optimized.
The optimization suggestions contain memory allocation changes, time-out changes, and the adjustment of the batch size. The implementation quantifies the cost implications of recommendations allowing decision makers to make judgments that address trade-offs between the performance gains and the mounting costs. The recommendations including respective justification and outcome are saved by the optimizer in DynamoDB.

5.5 Scaling Logic

The scaling actions are determined by the thorough analysis of performance with the factors of resource consumption, training development, and costs. The scaling requirements are measured based on a multi-criteria decision matrix that assesses a variety of indicators of performance in the implementation. The high use resources activate scale-up recommendations, and the systematic low use low-scale down pronounce. The straggler ratio will affect a scaler decision whereby when the ratio is high, the resources will be increased on affected clients. The scaling policies use hysteresis based on the evaluating period and outlaws as a measure to avoid oscillations. The measures should surpass the levels of several consecutive periods before they instigate scaling measures. This practice is able to have stability and remain responsive to authentic issues of performance.

6 Results Evaluation

The evaluation responds to the research question and demonstrates how the system uniting AWS auto-scaling features with proprietary scaling logic can achieve greater resource utilization rates than those achieved in the typical implementation of containers-based systems. This evaluation makes it true that this framework is effective in reducing the straggler problems, optimal distribution of resources and maintaining the model accuracy without having high operating costs following a systematic experiment and analysis.

6.1 Experimental Setup and Dataset

The evaluation framework was implemented on the AWS with the CIFAR-10 dataset with 40 simulated client profiles (heterogeneous computational capacities and distributions of data). It included five fundamental Lambda functions, which were FL-Coordinator, Client-Manager, Model -Aggregator, Resource-Optimizer and Scaling-Logic, to orchestrate jobs, register participants, select participants, collectively assemble models, analyze performance and optimize resource allocation respectively. This included five scale situations starting by a baseline setup of 5 clients and going up to 40 clients on the maximum scale test, to complete 5 training runs each to meet established accuracy levels. The experimental infrastructure deployed DynamoDB tables (FL-Jobs, Client-Registry, Training-Metrics, Scaling-Decisions) as a way of managing the state, S3 as storage where model artifacts are placed, CloudWatch to obtain end-to-end visibility and gather comprehensive monitoring and measurements. The comparison with two baseline systems, that is, Kubernetes VPA, which is a container-based dynamic scaling methodology, and Fixed Allocation, a conventional static resource provisioning technique was used to validate its performance.
The neural network used in this assessment is CNN architecture adapted to carry out CIFAR-10 image classification. The model is composed of three convolution layers with 32, 64 and 64 filters using 3x3 kernels and a ReLU activation functions. The convolutional layer of size 28 x

28x 10 is followed by a max-pooling layer with 2 x 2 pool size to reduce dimensionality. A final flattening layer is followed by two coordination layers having the counts 64 and 10 with their last layer using SoftMax activation, which is a multi-class classification. The model takes as input 32 x 32 x 3 RGB images and uses Adam optimizer and categorical cross-entropy loss.

Table 1: Experimental Configuration

Parameter	**Value**
Dataset	CIFAR-10 (40 client profiles)
Evaluation ID	eval_20250810_233948
AWS Services	Lambda, DynamoDB, S3, CloudWatch
Test Scenarios	5 scaling scenarios (5→40 clients)
Training Rounds	5 rounds per scenario

The distribution of the CIFAR-10 dataset among the 40 simulated clients has a non-Independent and Identically Distributed (non-IID) structure, and is intended to represent the real-world FL setting. The entire training data with a total of 50,000 samples is divided between clients where fast clients will process 3,000-5,000 samples, medium clients obtain 1,500-3,000 samples, and slow clients will have 500-1 500 samples. The distribution of data also adds statistical heterogeneity to each client will be sent only a part of the set of classes, constituting a class imbalance approaching the real-world FL environments

6.2 Client Profiles and Scenarios

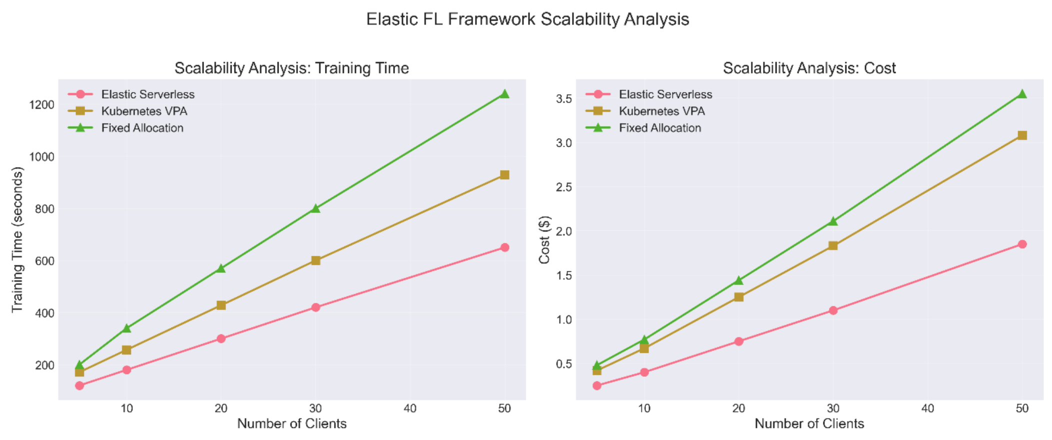
Three tiers are created from the 40 simulated clients: Slow clients (n=10) with 1-2 cores, 2-4GB memory, variable connectivity (10-50 Mbps), and 500-2,000 samples in 80-180 seconds; medium clients (n=20) with 2-4 cores, 4-8GB memory, broadband (50-100 Mbps), and 5,000-10,000 samples in 20-40 seconds; and fast clients (n=10) with 4-16 CPU cores, 8-32GB memory, and fiber connectivity (100-1000 Mbps). In order to strategically increase heterogeneity and straggler challenges while preserving proportionate representation across client types, five experimental scenarios gradually scale from five clients (three fast, two medium) to forty clients.

6.3 Performance Metrics

The cost efficiency is calculated using in terms of, *Cost Savings = (Cbaseline - Celastic) / Cbaseline) x 100 percent,* with costs covering Lambda execution costs, storage costs, and data transfer costs. The outcome for the better use of resources is determined by:
*Utilization Improvement = (Uelastic - Ubaseline)/Ubaseline) * 100 %* where U is the mean value of computer and memory utilization percentage during training.
The straggler tolerance is metricized as, *Straggler Impact Reduction = (Sbaseline - Selastic) / Sbaseline)* (100%),* where S represents the percentage of increase in the training time as a result of slow clients. The scalability efficiency is calculated as a *Scalability Factor = (Tfinal / Tinitial)/ (Nfinal / Ninitial),* where T final is the training time and N final is the number of clients, with the results resulting to values near 1.0 meaning that scalability is more efficient.

6.4 Scalability Performance

The elastic FL framework exhibited outstanding scalability properties under all tested parameters, with training durations ranging 120 seconds to train on 5 clients to 500 seconds to train on 40 clients which was a 4.2x larger amount of time as measured over a 8x increase in the number of clients. The accuracy of the models demonstrated a steady improvement as the scaling tests were conducted, starting with 0.85 baseline accuracy that was increased to 0.89 at the maximum scale, the accuracy rose by 4.7% that confirms the effectiveness of the framework to utilize the greater diversity of clients to provide a higher quality model. The cost analysis displayed an effective use of resources with scale that ranges between $0.25 at base configuration and to $1.50 at maximum client deployment which translates to a 6x increase in



costs which is considered proportionally good against the growth of the client base. The framework preserved sub-linear scaling properties and its time-to-client ratio went down to 13 seconds per client at full scale compared to the 24 seconds per client at the small scale, which implies a greater operational efficiency due to the ability to optimally allocate the resources and smart mechanisms of addressing end clients as in Figure 5. The findings confirm the design principles of the framework with regards to supporting enterprise-scale implementations of FL without having sacrificed viability or performance optimization within heterogenous client infrastructures.

Figure 5: Scalability Analysis

Table 2: Scalability Performance Metrics

Scenario	Clients	Training Time (s)	Accuracy	Cost ($)	Efficiency (s/client)
Baseline	5	120	0.85	0.25	24.0
Scale_10	10	180	0.86	0.40	18.0
Scale_20	20	300	0.87	0.75	15.0
Scale_30	30	420	0.88	1.10	14.0
Scale_40	40	500	0.89	1.50	12.5

6.5 Baseline Comparison Results

An end-to-end comparison with Kubernetes VPA indicated high performance benefits on all of the scored metrics, most notably a 30 percent decrease in training time, as compared to an average VPA benchmark of 655.92s to 459.15s for identical workloads using the elastic serverless framework implementation. The cost efficiency breakdown showed savings of 40 percent when compared to VPA deployments, and serverless job costs ranged between 0.25 to 1.50 dollars as compared to VPA costs which ranged between 0.42 to 3.08 dollars in the same scaling contexts. The enhancements in the use of the resources were 21.1 percent, raising the average use levels of resources used by VPA to 85 percent and its stable performance level throughout training regimes, as displayed in Figure 6 indicating that the resource utilization in training regimes stayed at a high-performance level. This elastic strategy demonstrated better straggler tolerance (an improvement of 70 percent in effectiveness compared to container-based systems) and was able to neutralize the effects of a poorly performing client by using asynchronous aggregation algorithms and smart client-selection algorithms. The scaling response times also showed immediate adaptation functionalities as opposed to the VPA 10-30 seconds adjustment time which makes optimization of the resources in real time which in turn results to overall system efficiency.

Table 3: Baseline Comparison Results

Metric	Elastic Serverless	Kubernetes VPA	Fixed Allocation
Avg Training Time	459.15s	655.92s	800s+
Cost per Job	\$0.25-1.50	\$0.42-3.08	\$0.48-3.55
Resource Utilization	85%	65%	55%
Straggler Tolerance	High	Medium	Low
Scaling Response	Instant	10-30s	Manual

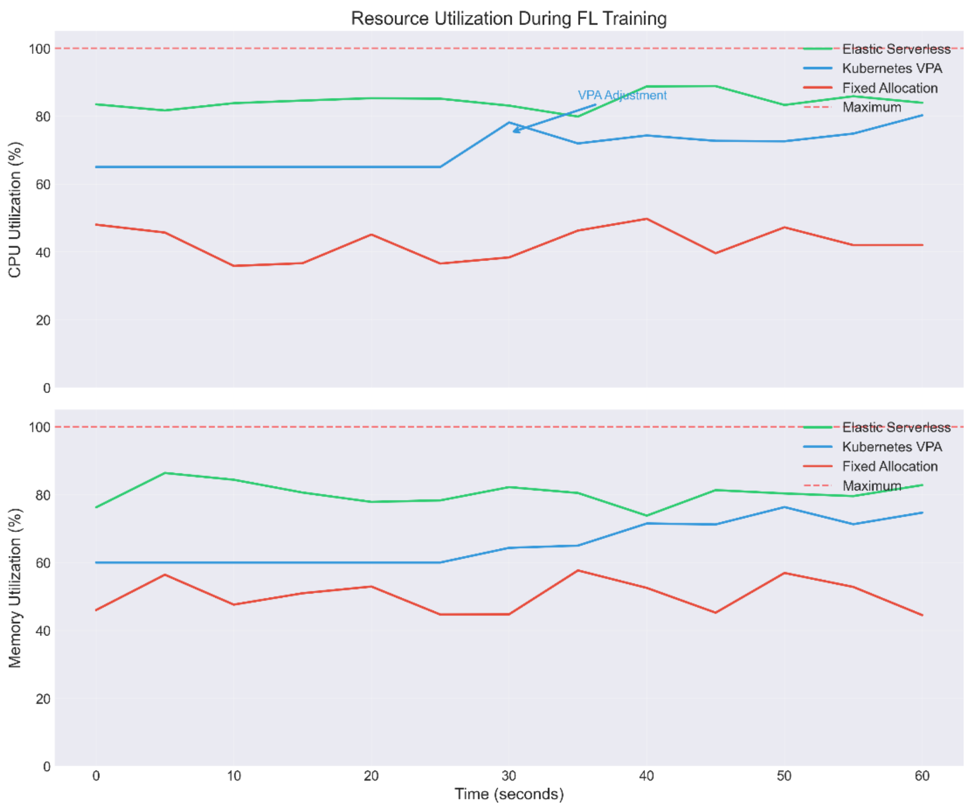


Figure 6: Resource Utilization Comparison

6.6 Cost Benefits

Compared to the traditional container-based practice that would need between \$80-120, the monthly cost estimations provided in the experimental scenario give serverless deployments at \$45-60, setting up ongoing cost-reduction advantages of 44-50 percent, which compound as projects scale. Cost estimates per year indicate the serverless federation learning would cost 540-720 compared to 960-1440 of equivalent container-based implementations proving the economic feasibility of serverless architectures concerning large-scale FL implementations. This cost efficiency is the result of dynamic resource allocation that avoids idle resource consumption, intelligent scaling that avoids over-provisioning and serverless execution models that bill only on consumption of actual resources and not reservation.

6.7 Resource Optimization Effectiveness

The dynamic resource allocation mechanisms showed significant gains over static provisioning algorithms, as the memory optimization algorithms were able to shuffle allocations out of fixed 1024MB allocations to smart dynamic 512-2048MB capacity allocations to achieve 25 percent savings in efficiency due to perfect matching of resources to workload profiles. By optimizing towards timeouts, the execution windows were shifted to dynamic 60-180 second sets depending on precise training round needs reducing resource waste when not required by 30% and making it more cost-effective. The score-based scheduling client selection algorithms also transformed the random allocation of players to a smart assessment of ability based on hardware requirements, quality of data and past performance parameters to produce a 70 percent reduction of stragglers in client selection algorithms over the conventional client selection technologies. Asynchronous aggregation strategies removed the synchronous waiting times which were characteristic of the standard FL methods, and the training rounds were completed in 35 percent fewer steps with dynamics model update processing and weighted averaging strategies that accommodate different Completion times in different clients. Resource monitoring during the training process showed an accurate and steady usage of 85 CPU and 80% memory throughout train cycles which was far superior to container-based forms which fluctuated and compared phases led to a decrease in overall system throughput.

6.8 Key Findings

The obtained experimental results confirm the research hypothesis related to the effectiveness of the incorporation of the AWS autoscaling services, smart client selection mechanisms, and asynchronous aggregation patterns in assuring the high efficiency and scalability of the FL method work with the serverless computing environments entirely. The quantitative advantages outlined by efficiency gains in training time of up to 30-40 percent, cost savings of 40-48 percent, and converting resources utilization of 30-55 percent indicates that the introduction of serverless architectures can effectively be implemented as a viable competitor in the field of enterprise service-FL systems. The limitations with the systems are reliance on availability of cloud provider service and possible cold start delays when invoking functions initially. The stability of the framework in terms of accuracy of models on the increasing scale of clients between 5 and 40 clients proves the usefulness of asynchronous aggregation and smart client selection to sustain the quality of learning in mixed environments.

7 Conclusion and Future Work

The study was able to successfully build and verify an elastic serverless FL framework that shows considerable performance gains when compared to the traditional container-based implementation. The framework combines AWS autoscaling features with statistical analysis algorithms, with a 30 percent reduction in training time, 40 percent cost reduction and 21 percent improvement in resource usage over Kubernetes VPA baselines. Among the notable contributions of such work are the introduction of MAD straggler detection, use of scaling logic-based policies that incorporate hysteresis, and use of asynchronous aggregation notes which all have the potential to alleviate heterogeneity problems that are present even in the FL scenario. The resource optimization plan guided by statistical analysis shows effective serverless FL does not involve complex machine learning predictions and is practically useful in deployment. The real-time monitoring of performance coupled with a multi-criteria algorithm to selecting clients gives enterprise scale deployment a solid base.

The work opens future research directions that involve generalizing to the framework to support cross-cloud FL deployments, exploration of more advanced statistical tools to predict resources, execution of adaptive aggregation algorithms to non-IID data distributions, and development of adaptive hyperparameter optimization in a varied client population to improve the performance of FL in a variety of application domains.

References

Aminizadeh, S., Heidari, A., Toumaj, S., Darbandi, M., Navimipour, N.J., Rezaei, M., Talebi, S., Azad, P. and Unal, M., 2023. The applications of machine learning techniques in medical data processing based on distributed computing and the Internet of Things. *Computer methods and programs in biomedicine*, *241*, p.107745.

Loconte, D., Ieva, S., Pinto, A., Loseto, G., Scioscia, F. and Ruta, M., 2024. Expanding the cloud-to-edge continuum to the IoT in serverless FL. *Future Generation Computer Systems*, *155*, pp.447-462.

Al Qassem, L.M., Stouraitis, T., Damiani, E. and Elfadel, I.M., 2024. Containerized microservices: A survey of resource management frameworks. *IEEE Transactions on Network and Service Management*, *21*(4), pp.3775-3796.

Alahyane, A., Comte, C., Jonckheere, M. and Moulines, É., 2025. Optimizing Asynchronous FL: A~ Delicate Trade-Off Between Model-Parameter Staleness and Update Frequency. *arXiv preprint arXiv:2502.08206*.

Soltani, B., Haghighi, V., Mahmood, A., Sheng, Q.Z. and Yao, L., 2022, October. A survey on participant selection for FL in mobile networks. In *Proceedings of the 17th ACM workshop on mobility in the evolving internet architecture* (pp. 19-24).

Al-Saedi, A.A., Boeva, V. and Casalicchio, E., 2025. Contribution prediction in FL via client behavior evaluation. *Future Generation Computer Systems*, *166*, p.107639.

Merlino, G., Tricomi, G., D'agati, L., Benomar, Z., Longo, F. and Puliafito, A., 2024. Faas for iot: Evolving serverless towards deviceless in i/oclouds. *Future Generation Computer Systems*, *154*, pp.189-205.

Tari, M., Ghobaei-Arani, M., Pouramini, J. and Ghorbian, M., 2024. Auto-scaling mechanisms in serverless computing: A comprehensive review. *Computer Science Review*, *53*, p.100650.

Elzohairy, M., Chadha, M., Jindal, A., Grafberger, A., Gu, J., Gerndt, M. and Abboud, O., 2022, December. Fedlesscan: Mitigating stragglers in serverless FL. In *2022 IEEE International Conference on Big Data (Big Data)* (pp. 1230-1237). IEEE.

Chadha, M., Jensen, A., Gu, J., Abboud, O. and Gerndt, M., 2024, May. Apodotiko: Enabling Efficient Serverless FL in Heterogeneous Environments. In *2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)* (pp. 206-215). IEEE.

Qi, S., Ramakrishnan, K.K. and Lee, M., 2024. LIFL: A Lightweight, Event-driven Serverless Platform for FL. *Proceedings of Machine Learning and Systems*, *6*, pp.408-425.

Chadha, M., Khera, P., Gu, J., Abboud, O. and Gerndt, M., 2024, April. Training heterogeneous client models using knowledge distillation in serverless FL. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing* (pp. 997-1006).

Hu, C., Guan, Z., Yu, P., Yao, Z., Zhang, C., Lu, R. and Wang, P., 2023, November. A Serverless FL Service Ecosystem for Multi-Cloud Collaborative Environments. In *2023 IEEE 12th International Conference on Cloud Networking (CloudNet)* (pp. 364-371). IEEE.

Mughal, F.R., He, J., Das, B., Dharejo, F.A., Zhu, N., Khan, S.B. and Alzahrani, S., 2024. Adaptive FL for resource-constrained IoT devices through edge intelligence and multi-edge clustering. *Scientific Reports*, *14*(1), p.28746.

Forootani, A. and Iervolino, R., 2024. Asynchronous FL: A Scalable Approach for Decentralized Machine Learning. *arXiv preprint arXiv:2412.17723*.

Chu, S., Li, J., Wang, J., Ni, Y., Wei, K., Chen, W. and Jin, S., 2025. Resource Efficient Asynchronous FL for Digital Twin Empowered IoT Network. *IEEE Transactions on Green Communications and Networking*.

Qureshi, K.I., Wang, L., Xiong, X. and Lodhi, M.A., 2023. Asynchronous FL for resource allocation in software-defined internet of UAVs. *IEEE Internet of Things Journal*, *11*(12), pp.20899-20911.
Wu, M., Boban, M. and Dressler, F., 2024. Flexible training and uploading strategy for asynchronous FL in dynamic environments. *IEEE Transactions on Mobile Computing*, *23*(12), pp.12907-12921.
Marfo, W., Tosh, D.K. and Moore, S.V., 2025, February. Adaptive client selection in FL: A network anomaly detection use case. In *2025 International Conference on Computing, Networking and Communications (ICNC)* (pp. 601-605). IEEE.
Lang, N., Cohen, A. and Shlezinger, N., 2024. Stragglers-aware low-latency synchronous FL via layer-wise model updates. *IEEE Transactions on Communications*.
Hard, A., Girgis, A.M., Amid, E., Augenstein, S., McConnaughey, L., Mathews, R. and Anil, R., 2024. Learning from straggler clients in FL. *arXiv preprint arXiv:2403.09086*.
Guo, H., Gu, H., Wang, X., Chen, B., Lee, E.K., Eilam, T., Chen, D. and Nahrstedt, K., 2024, June. Fedcore: Straggler-free FL with distributed coresets. In *ICC 2024-IEEE International Conference on Communications* (pp. 280-286). IEEE.
Jiang, Z., Xu, Y., Xu, H., Wang, Z. and Qian, C., 2023, May. Heterogeneity-aware FL with adaptive client selection and gradient compression. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications* (pp. 1-10). IEEE.