Intelligent Framework in a Serverless Computing for Serving using Artificial Intelligence and Machine Learning

Deepak Khatri[1], Sunil Kumar Khatri[2], Deepti Mishra[3]
AIIT, Amity University, Noida, India[1, 2]
NTNU, Norway[3]

***Abstract*—Serverless computing has grown in popularity as a paradigm for deploying applications in the cloud due to its ability to scale, cost-effectiveness, and simplified infrastructure management. Serverless architectures can benefit AI and Machine Learning (ML) models, which are becoming increasingly complex and resource-intensive. This study investigates the integration of AI/ML frameworks and models into serverless computing environments. It explains the steps involved, including model training, deployment, packaging, function implementation, and inference. Serverless platforms' auto-scaling capabilities allow for seamless handling of varying workloads, while built-in monitoring and logging features ensure effective management. Continuous integration and deployment pipelines simplify the deployment process. Using serverless computing for AI/ML models offers developers scalability, flexibility, and cost savings, allowing them to focus on model development rather than infrastructure issues. The proposed model leverages performance forecasting and serverless computing model deployment using virtual machines, specifically utilizing the Knative platform. Experimental validation demonstrates that the model effectively predicts performance based on specific parameters with minimal data collection. The results indicate significant improvements in scalability and cost efficiency while maintaining optimal performance. This performance model can guide application owners in selecting the best configurations for varying workloads and assist serverless providers in setting adaptive defaults for target value configurations.**

***Keywords*—Machine learning; data analytics; serverless computing; performance testing**

I. Introduction

A cloud computing architecture called "serverless computing" uses dynamic resource management and allocation by the cloud provider to run and scale applications. Developers use this paradigm to build and distribute code in the form of brief, stateless functions, while the cloud provider takes care of infrastructure management tasks including server provisioning, scalability, and maintenance. In traditional computing models, developers are responsible for managing servers and infrastructure resources, which can be time-consuming and require expertise in managing scalability and availability. Developers can concentrate entirely on building and deploying code thanks to serverless computing, which abstracts away the infrastructure layer. There are some key characteristics of serverless computing, which include:

- Event-driven execution: Serverless functions are triggered by events, such as HTTP requests, database updates, or message queue events. Functions are executed on-demand in response to these events.
- Scalability: Serverless platforms automatically scale the number of instances running the functions based on the incoming workload. Scaling is performed transparently, without developers needing to provision or manage additional servers.
- Pay-per-use billing: With serverless computing, developers are billed based on the actual usage of their functions. Cloud service providers are not charging for idle resources, which makes it cost-efficient for applications with variable or sporadic workloads.
- Stateless functions: Serverless functions are designed to be stateless, meaning they do not maintain any internal state between invocations. Any required state information is typically stored in external data stores, such as databases or object storage.

There are several benefits of serverless computing, that includes reduction of operational overheads, automatic scaling, reduction of cost, and increased flexibility. Developers focused on writing code rather than managing servers, operating systems, or scaling mechanisms. This allows for faster development cycles and increased productivity. Serverless platforms handle the scaling of functions automatically, ensuring that applications can handle varying workloads without the need for manual intervention. Serverless computing eliminates the cost of idle resources and pays only for actual function execution. This makes serverless computing cost-effective for applications with unpredictable or low usage patterns. Serverless functions are often platform-agnostic, can be written in various programming languages, and can integrate with other cloud services, offering developers a wide range of functionalities.

Serverless computing has gained popularity for a variety of use cases, including web and mobile backends, data processing,

IoT applications, and microservice architectures. It offers developers a scalable and cost-effective way to deploy applications without the burden of managing the underlying infrastructure [2]. While serverless computing offers several benefits, there are also challenges associated with adopting and implementing this paradigm. Here are some common challenges in serverless computing:

- Cold Start Latency: When a function is called for the first time or after a period of inactivity, serverless functions have an inherent cold start latency. This is because the cloud provider needs to provision and initialise the necessary resources to execute the function. Cold start latency can impact real-time or low-latency applications that require immediate response times.
- Limited Execution Time: Serverless platforms often impose execution time limits on functions, typically ranging from a few seconds to a few minutes. Long running or computationally intensive tasks may face challenges in fitting within these constraints. In such cases, alternative architectures or breaking tasks into smaller functions may be required.
- Vendor Lock-in: Serverless platforms may have proprietary interfaces, service contracts, and vendor specific features. Migrating serverless functions between different cloud providers can be complex and time-consuming, potentially leading to vendor lock-in. Careful consideration and abstraction of vendor-specific functionality can mitigate this challenge.
- Monitoring and Debugging: Debugging and monitoring serverless functions can be more challenging compared to traditional architectures. Fine-grained logging, tracing, and performance monitoring tools are crucial for identifying and diagnosing issues within serverless functions. However, some platforms have limitations in terms of logging granularity and debugging capabilities.
- Resource Limitations: Serverless platforms impose resource limits, such as memory allocation, CPU usage, and storage. Applications with resource-intensive workloads, such as large-scale data processing or AI/ML models, may encounter restrictions that require careful optimisation and scaling considerations.
- State Management: Serverless functions are designed to be stateless, which means it do not maintain internal state between invocations. While this simplifies scalability, it can pose challenges for applications that require maintaining session or contextual data. External storage or database services must be utilised to manage and retrieve state information.
- Testing and Local Development: Developing and testing serverless functions locally can be challenging due to the need for specific platform emulation or integration with cloud services. Local development environments often lack the same operational characteristics as the serverless platform, making it difficult to reproduce certain behaviours.
- Security and Compliance: Serverless computing introduces new security considerations. Function isolation, access control, and secure integration with other services must be carefully addressed. Compliance with regulations and data privacy requirements may also present challenges when handling sensitive data in a serverless environment.

While these challenges exist, many can be mitigated with careful architectural design, a proper understanding of platform limitations, and the utilisation of supporting tools and services. As serverless computing continues to evolve, cloud providers are addressing these challenges and providing improved capabilities and tooling for developers. Although serverless computing provides several benefits compared to cloud computing services. However, an intelligent framework can leverage AI and ML techniques to analyse historical usage patterns, workload characteristics, and performance metrics to optimise auto-scaling algorithms. By accurately predicting resource demands, the framework can ensure efficient scaling, minimising the occurrence of underutilised or overburdened resources.

An intelligent framework can dynamically allocate requests based on factors like function availability, resource utilisation, and latency. By intelligently routing traffic, it can optimise resource utilisation and improve overall performance. An intelligent framework can intelligently orchestrate workloads based on their characteristics, such as prioritising latency-sensitive tasks, distributing compute intensive tasks across available resources, or dynamically adjusting resource allocation based on workload dynamics. An intelligent framework can analyse usage patterns, pricing models, and optimisation algorithms to minimise costs while meeting application requirements. It can recommend optimal function configurations, memory allocations, or scaling strategies to optimise cost-effectiveness. By incorporating intelligent features, an intelligent framework can enhance the performance, efficiency, scalability, and cost-effectiveness of serverless computing environments [2]. It can automate complex decision-making processes, optimise resource allocation, and improve the overall user experience, making it easier for developers to harness the benefits of serverless computing while minimising the associated challenges.

Adaptive Function Placement (AFP) is one of the critical factors that refers to the process of dynamically assignment of serverless functions to appropriate computing resources based on real-time workload demands and system conditions. AFP techniques consider various factors when determining the placement of functions, such as workload characteristics, resource availability, and performance objectives. There are several benefits of AFP, such as:

- Workload Monitoring: Monitoring the workload characteristics is crucial for effective function placement. This involves collecting data on factors like request rate, latency, resource utilisation, and network conditions. Real-time monitoring enables the system to adapt to changing workload patterns.

- Resource Availability: The AFP system needs to be aware of the available computing resources in the serverless environment. This includes information about CPU capacity, memory, network bandwidth, and other relevant resource metrics.
- Load Balancing: Load balancing is an important aspect of AFP. It involves distributing the workload evenly across available resources to prevent resource bottlenecks and ensure efficient resource utilization. Load balancing algorithms consider factors like function size, resource requirements, and current resource utilisation to make informed placement decisions.
- Cost Optimisation: AFP techniques often aim to minimise costs by dynamically allocating resources based on demand. By monitoring workload patterns and resource usage, the system can make decisions that optimise cost efficiency, such as scaling down resources during low demand periods and dynamically scaling up during peak loads [1].
- Latency and Performance: AFP also considers the latency and performance requirements of functions. By analysing factors like network latency, function dependencies, and data locality, the system can place functions closer to the data sources or reduce network hops, thereby reducing latency and improving overall performance.
- Dynamic Scaling: AFP techniques often involve the dynamic scaling of resources based on workload demand. This includes automatically provisioning additional resources when the workload increases and releasing them when the demand decreases. Dynamic scaling ensures optimal resource allocation and responsiveness to varying workloads.

The major contribution of the current research is as follows:

- The proposed model can perform a large degree of parallelism in a large-scale system.
- The proposed model improves the performance parameters with response time and cost.
- The presented model has inherent features of performance, cost, and distinct workloads.

II. Literature Review

There have been several research projects in the past for the design and implementation of frameworks for serverless computing. The viability of employing a serverless architecture for AI workloads was investigated by Ishakian et al. It was evaluated for the effectiveness of providing serverless deep learning functions that categorise images by running the model via a forward pass [8]. The data shows that warm serverless function executions have a reasonable latency, but cold starts have a considerable cost. Adherence to SLAs that do not account for this bimodal latency distribution may be in jeopardy. Because functions are stateless and serverless frameworks lack access to GPUs, each function execution can only consume CPU resources, and performance cannot be enhanced by depending on the serverless platform runtime to store state between invocations.

The Function-as-a-Service paradigm, in which users create brief functions that are subsequently managed by a cloud platform, as illustrated by Castro et al. The approach has several applications, including big-data analytics, event handlers, and bursty invocation patterns. By giving the platform provider a major portion of the operational complexity of monitoring and expanding large-scale applications, serverless computing lowers the bar for developers [6]. The developer must now overcome constraints imposed by the statelessness of their functions and comprehend how to relate the SLAs of their application to those of the serverless platform and other reliant services.

Workload profiling with benchmarks was used by Lioyd et al. to analyse the specific resource needs of very diverse workloads and anticipate the cost of workloads in various situations. Their research presupposed a fixed environment with a uniform VM capacity and initial configurations. The workload capacity and resource utilisation of the servers are quite dynamic while managing serverless architecture, nevertheless. A Cloud-Scale Java profiler was created by Yin et al. to help developers identify performance-related issues with their applications. It also gave developers insight into the system's throughput and the resources that each microservice would need to reach a certain level of service quality. Ye et al. employed profiling and normalised performance to increase workload performance and predict the influence of currently running VMs and co-location. It was suggested as a technique that, by utilising VM migration, improves workload performance while lowering PM energy usage [14], [18]. Even though they make some intriguing points, their algorithm is incompatible with a wide range of workloads.

By adjusting task designs and resource allocation choices, Li et al. optimise the performance levels of composite service application activities using analytical models based on queuing theory [10]. For capacity analysis and profiling of multitier internet server applications, Apte et al. suggested a load-generating tool. Their effort aims to produce a thorough profile of server resource utilisation, broken down by request type [3]. A multi-objective optimisation was used by Liu et al. to locate the ideal location for containers. However, it was assumed that there was only one application running in the cluster while considering the nodes' varying runtime environments. Additionally, because they make no generalisations, they must carry out the optimisation for each application separately [10], [11].

Kaffes et al. presented distinct serverless computing platforms, such as centralised schedulers and core-granular, that can be utilised without infrastructure [9]. The authors contend that distinct characteristics of serverless computing platforms include burstiness, brief and unpredictable execution times, statelessness, and single-core execution. Additionally, according to their research, which is supported by Wang et al., the scalability of present serverless products is inefficient [17]. Bortolini et al. conducted experiments with a variety of setups and FaaS providers to identify the key variables affecting the

performance and price of the most recent serverless systems [5]. It was discovered that the programming language being utilised is one of the most crucial elements for both performance and cost. Additionally, they identified one of serverless computing's biggest shortcomings as low-cost predictability. Lloyd et al. are investigating the effectiveness and performance of serverless computing platforms [12], [13]. Bardsley et al. evaluated the performance of AWS Lambda in terms of distinct factors such as availability, low-latency, and infrastructure management. The authors found that infrastructure is not visible to the end-user and provides a better interface, which underlies the fundamental concepts [4].

Hellerstein et al. addressed the main faults and antipatterns in the first-generation serverless computing platforms. The author shows the implementation details and distributed computing platform that has cloud-based applications [7]. There are some issues with the current approach, such as the absence of global states and lambda function inability. The key issues inhibiting the widespread adoption of FaaS, according to Eyk et al., are significant overheads, variable performance, and new sorts of cost-performance trade-offs [15]. A strategy was developed to address six performance-related issues facing the serverless computing sector in their work. According to Zheng et al.'s, the performance of distinct platforms depends on the workload, implementation of the FaaS system, and the optimal set of parameters. Table I shows the comparative study of the current research with the existing work, as demonstrated in the result section with better outcomes.

TABLE I. COMPARATIVE ANALYSIS

References	Average Concurrency	Response Time	Time Delay	Average Containers
Yin et al.	2.6	10.2	0.02	5
Hellerstein et al.	2.8	11.5	0.12	6
Zheng et al.	3.2	11.8	0.25	5
Kaffes et al.	2.5	10.9	0.11	4
Lloyd et al.	3.0	10.7	0.21	5
Liu et al.	3.1	10.5	0.19	6
Proposed Work	3.4	10.1	0.01	6

After a rigorous literature review, it has been found that there are some gaps in the field of serverless computing where several new research projects can be proposed with critical investigation. The author has tried to fill the gap by building an intelligent system that has a large degree of parallelism on a large scale. In the next section, it has defined as a proposed methodology for the achievement of the objectives of the current research.

III. PROPOSED METHODOLOGY

The importance of machine learning in a serverless computing environment involves combining various technologies to create scalable, efficient, and intelligent systems. Machine learning models within containers can facilitate easy deployment and management in a serverless environment. An intelligent framework refers to the dynamic design of serverless computing. It includes several advancements for the best utilisation of the resources on the server side. Adaptive Function Placement (AFP) is one of the critical factors that refers to the process of dynamically assigning serverless functions to appropriate computing resources based on real-time workload demands and system conditions [16]. AFP aims to optimise resource allocation, maximise performance, and minimise costs in serverless computing environments. Applications are created and deployed using serverless computing as functions that are called when certain events or requests occur. The developer is abstracted away from the underlying infrastructure and resource management, and these operations are carried out in a managed environment provided by the cloud service provider. Statistical machine learning is used in this study to create and examine the placement of an adaptive function that serverless computing systems can use to improve running function performance while lowering operating costs. The suggested adaptive function placement technique can be simply implemented by using container orchestration in the case of serverless computing providers. It also affects the distinct findings and is utilised to incorporate them with issues generated during the implementation phases. The system is implemented using Knative scale platform (Castro et al., 2019).

Fig. 1 shows the Knative scale calculation open-source platform. Knative is an open-source platform built on top of Kubernetes that provides a set of building blocks for creating modern, source-centric, and container-based applications. It abstracts away the complexities of managing containerised workloads, auto-scaling, and event-driven architectures. One of the key features of Knative is its ability to automatically scale applications based on incoming traffic. Knative allows you to define rules and thresholds for scaling your application. For example, you can set thresholds for CPU usage or request throughput that, when exceeded, trigger scaling actions. The Knative Scale Calculation module is responsible for determining how to scale your application based on incoming traffic and load. The Knative scale can be divided into distinct modules, such as:

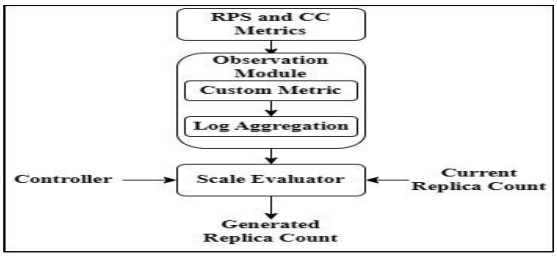

Fig. 1. Knative scale calculation.

A. Metrics

Two metrics, named Requests per Second (RPS) and Concurrency Value (CC), are the metrics that can be used for driving auto-scaling in a metric-based approach. The concurrency value represents the number of active requests that are being concurrently processed by an instance of your application. It is a measure of how effectively the application handles multiple requests at the same time. As the CC value increases, it suggests that the application is under a higher concurrent load, potentially necessitating auto-scaling to ensure

optimal performance. When the CC value drops, auto-scaling can reduce the number of instances to match the lower concurrency level. The RPS metric provides insight into the workload and demand on your application. As the RPS increases, it indicates higher user activity, and auto-scaling can be triggered to accommodate the increased load by deploying more instances of your application. Conversely, if the RPS decreases, auto-scaling can reduce the number of instances to save resources.

B. Observation Module

Knative can integrate with external observability tools like Prometheus, which is a popular monitoring and alerting toolkit. These tools help collect and store the metrics generated by the application and infrastructure. There are several key aspects to the observation module in Knative.

- Metrics Collection: Knative can leverage Kubernetes metrics for monitoring and scaling decisions. Kubernetes provides built-in metrics like CPU usage, memory usage, and request throughput.
- Autoscaling Metrics: As mentioned earlier, Knative can use metrics like Requests per Second (RPS) and Concurrency Value (CC) for autoscaling decisions. These metrics help determine the current load on the system and adjust the number of instances accordingly.
- External Monitoring Tools: While not inherently a part of Knative itself, observability tools like Prometheus, Grafana, and others can be integrated with Knative to provide comprehensive monitoring and visualisation of metrics.
- Application Tracing: Observability often includes application tracing to understand how requests flow through the system and identify bottlenecks or issues. Tools like Jaeger can be integrated to provide distributed tracing capabilities.
- Log Aggregation: Effective observation also involves collecting and aggregating logs from various components of the system. Centralised log management tools like Elasticsearch, Fluentd, and the Kibana (EFK) stack can be used for this purpose.

Event Streaming: Since Knative is event-driven, monitoring and observing events becomes important. Event streaming platforms like Apache Kafka can be integrated to manage and analyse events.

Custom Metrics: Depending on the application's requirements, custom metrics might be needed. Knative supports the use of custom metrics to make scaling decisions that align with the specific needs of the application. In order to prevent making rash conclusions while evaluating scaling, the purpose of this module is to produce steady observations.

C. Scale Evaluator

The scale evaluator generates the order for the new replica count by utilising the observed values and the current replica count. The current replica is generated by monitoring the distinct FPS or CC. By default, the Kubernetes deployment's new replica target is set by the Knative auto-scaling evaluation, which occurs every Teva (2 seconds in Knative). Eq. (1) is used to evaluate the generated replica by using the observed value and the current replica. The observed values are the values of the metric in terms of RPS or CC.

$$Generated_{Replica} = \frac{Observed\_value}{Current\_Replica} \quad (1)$$

The suggested system can be divided into distinct modules.

A high-level view of the suggested performance model is shown in Fig. 2. The metric module is responsible for collecting, processing, and analysing various metrics and data to monitor and assess the performance, health, and behaviour of the system. It is utilised to evaluate the observed module distribution with the help of the evaluator module. These parameters are being evaluated by using the CC and RPS with the arrival rate. The average request arrival time was provided by the input.

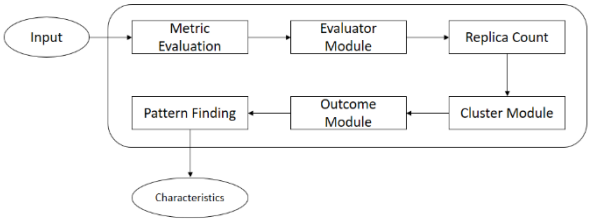

Fig. 2. Proposed methodology.

This step is crucial because it captures several crucial aspects, including the amount of work and distribution time needed for the deployment of the setup. The memory utilisation and CPU time are also evaluated for the generation replica count. The replica count is done based on the total values of the target variable and the attributes generated during the processing. The Cluster Module would be responsible for managing the Kubernetes cluster on which Knative applications are deployed. This includes provisioning, scaling, and maintaining the nodes that form the cluster. The probabilities of the cluster module can be described in Eq. (2).

$$Pr(x,y)\ (x',y') = Pr(x,x')(y) * Pr(y,y')(x') \quad (2)$$

where, $Pr_{(x,y)\ (x',y')}$ defines the transitioning probabilities from the current state to the transmission of rows to columns and vice versa. An output module refers to the final trained model that generates predictions or classifications based on input data. The evaluation of replica count would be done by using the outcome module and generating a model for similar patterns. After evaluation of the patterns, the likelihood of the replica count can be evaluated in terms of the similar characteristics. Finally, the ready container is utilised for the output and computation of the performance of the system. The average replica count can be written in terms of Eq. (3).

$$N' = \sum_{i=1}^{n} N_i\, \alpha_i \quad (3)$$

where, N' is the average number of replicas count that can be evaluated by using the current replica count and a constant factor of the observed value.

D. Experimental Testbed Setup

For the experimental testbed, a virtual machine (VM) hypervisor is setup, which is utilised for the data collection. The proposed system has been implemented with the following configuration: The processor is of the 7th generation of the Intel series, with 2 TB of HDD, 2 GB of RAM, and VMWARE 15.5.1. Four nodes served as worker nodes on the Cybera cloud, while RabbitMQ served as our distributed task queue. Kubernetes version 1.20.0 is used for our cluster, along with Kubernetes client (kubectl) version 1.18.0 and Python 3.8.5 for the customer. Profiling and performance measurement are two separate phases that might be divided into the data collection phase, depending on the situation. The primary objectives of the profiling phase of data gathering are to characterise the workload and identify any unique requirements. So, a dedicated VM is used to host the container that needs to be profiled while measuring the throughput on the client side and the various resource utilisation statistics shown in Table II.

TABLE II. STATISTICS FOR RESOURCE UTILIZATION

Variable	Units	Remarks
vCPU	4	Setup a network
Latency	Less than 1 ms	Minimum time delay
OS	VM Ubuntu	Virtual
Network	10Mbps	Fast response

The evaluation of resources for VM is providing the impact on the container in data collection. To do this, haphazardly distributed sets of containers on a virtual machine are setup, each of which produces a haphazard workload. Before deploying the new container, we next evaluate how much of the available resources are being used by this erratic workload. The achieved performance is then tracked using Eq. 4, and all the results are maintained in the data set for the predictive performance model. A training set of 128 of the 183 data points was collected for the model used in the trials, and a test set of 55 was used.

$$T_N = {}^{T_F}\!/_{T_p} \tag{4}$$

where, T_N is the normalized throughput used to evaluated by the division of T_F function-based throughput and T_P generated during the profiling phase.

E. Machine Learning Module

For this work, a variety of data-driven modelling strategies have been examined. The versatility, ability to fit nonlinear functions, and minimal computing costs of artificial neural networks built on TensorFlow were our preferred options in this case. To find out how effective this strategy is, in-depth experimental tests were conducted. Various machine learning algorithms were analysed for predicting the normalised throughput of the serverless platform in order to construct the predictive performance model. Among the methods employed are artificial neural networks, decision tree regression, random forest regression, support vector regression, and linear regression. The system's container performance (i.e., throughput and reaction time) fluctuates nonlinearly based on the workload characteristics. As a result, it is expected that linear models (linear regression) will perform poorly when compared to nonlinear techniques. In our experiments, we found that SVR and neural networks had the best accuracy performance, with neural networks marginally surpassing SVR. Neural networks were chosen in this study for our tests due to their generality, flexibility, adaptability, and prediction speed to fit nonlinear functions. Table III contains the neural network setup that was employed.

TABLE III. MACHINE LEARNING CONFIGURATION

Functions	Size
ReLu	0.0 to 1.0
Convolutional Net	5*5
Activation Map	300*200 pixels

F. Optimised Algorithm

The major aim of the proposed scenario is to recognise the unique features during the execution of the VM container. Those unique features of a workload are based on the resource usage of the container on a VM. It is difficult to assess the performance decrease caused by collocating with another container because of the extreme diversity of workloads on such platforms. By creating a predictive performance model that analyses each workload and forecasts its normalised performance when deployed to a particular VM, it was attempted to get around this limitation. Finding the VM that has the least detrimental effect on the performance of the container is the answer to the question of which virtual machine is best to deploy the container on. To accomplish this, it is suggested that a fast-profiling step be added to the serverless platform during the container installation process. This phase will give a sample workload that the user has specified. When scaling a function after the profiling process, the profile is utilised for evaluating the performance of the VM functions and the prediction model to assess how effectively each VM is using its resources.

G. Testing and Validation

Testing and validation using machine learning involves applying various techniques and methodologies to assess the performance, accuracy, and generalisation capabilities of machine learning models. Proper testing and validation are crucial to ensuring that machine learning models work well on unseen data and provide reliable predictions or classifications. Once the model is fine-tuned using the validation set, it is evaluated on the test set, which should represent unseen data. This provides an unbiased estimate of the model's real-world performance.

IV. RESULTS AND DISCUSSION

The measured and anticipated average number of containers that are ready to meet incoming requests are shown for various setups in Fig. 3 and Fig. 4. Here, the deployment cost is represented by the typical number of containers. Depending on the setup, the deployment's cost may be VM-based in a Kubernetes cluster or pod-based in a Google Cloud Run deployment. The expenses of the infrastructure, however, will be inversely correlated with the typical number of containers in both cases. The average concurrency value for various settings

is shown in Fig. 5 and Fig. 6, respectively. These parameters can help the developer accurately configure other services on which the deployment depends. As an illustration, the capacity provided by most managed database solutions may be configured to maximise performance while minimising expenses. For this deployment, the Quality of Service (QoS) metric has been the average response time. The measured and anticipated average response times for various configurations and arrival rates are shown in Fig. 7 and Fig. 8, respectively. In contrast to the predetermined arrival rate, the average number of containers available to fulfil requests in our studies has varied goal concurrency values. As you can see, the scale on the x-axis is logarithmic. The vertical bar shows the 95% confidence intervals, which in this case were relatively small because the experiments lasted long enough to produce highly dependable results.

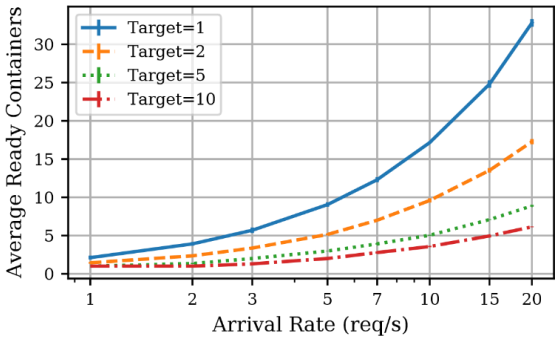

Fig. 3. Average number of containers.

In Fig. 3, there is a description of goal concurrency that can be evaluated by using the average number of containers and packet arrival rate. The x-axis represents the values on a logarithmic scale, while the y-axis represents the distinct targets for containers. Autoscaling policies, often based on metrics like Requests per Second (RPS) or Concurrency Value (CC), work in tandem with desired concurrency values. When the observed concurrency exceeds the desired value, scaling policies can trigger the necessary scaling actions.

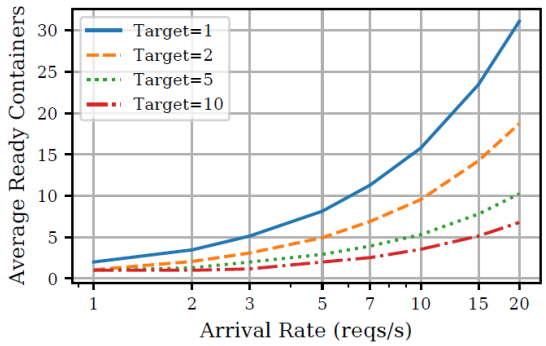

Fig. 4. Desired concurrency values.

Desired concurrency values typically refer to the target level of simultaneous requests or tasks that an application or system aims to maintain. In the context of auto-scaling and performance optimisation, determining the appropriate desired concurrency values is crucial for achieving optimal resource utilisation and user experience. Fig. 4 shows the concurrency values in terms of average containers vs. fixed arrival rate. The x-axis represents the values on a logarithmic scale, while the y-axis represents the distinct average-ready containers.

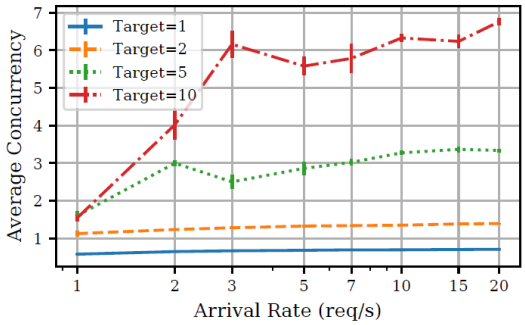

Fig. 5. Target concurrency values.

Target concurrency values typically refer to the specific levels of concurrent requests that an application or system aims to achieve under various conditions. These values help guide scaling behaviour and resource allocation in order to maintain optimal performance and responsiveness. Fig. 5 shows the graph between average concurrency and fixed arrival rate on a logarithmic scale.

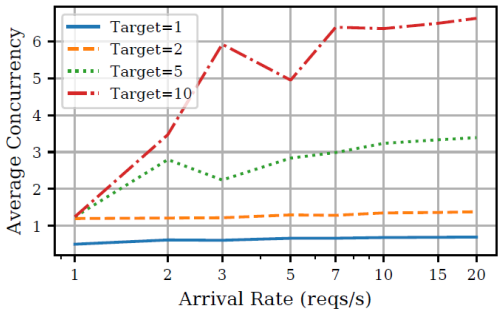

Fig. 6. Predicted concurrency values.

Predicted average concurrency refers to the estimated or forecasted level of concurrent requests that an application or system is expected to experience over a specific period. This prediction is typically based on historical data, patterns, trends, and potentially external factors that influence the demand for the application. Fig. 6 shows the predicted average concurrency vs. fixed arrival rate on the x-axis.

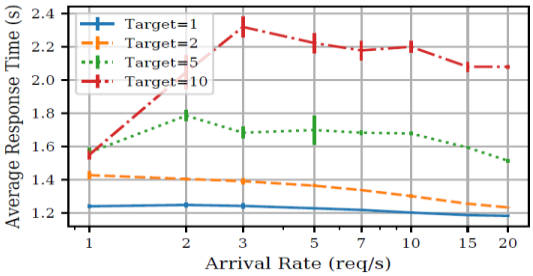

Fig. 7. Response time vs. Arrival time.

Average response time and fixed arrival rate are two important concepts in performance analysis and capacity planning for systems, including serverless architectures like

Knative. Average response time, also known as average latency, is the time it takes for a system to respond to a request on average. It's a critical metric for assessing the performance and user experience of an application. Lower average response times generally indicate better system performance and faster user interactions. In the context of Knative, average response time is influenced by factors such as the processing time of requests, network latency, resource availability, and system architecture. Monitoring and optimising average response times are essential to ensuring that users experience responsive and efficient applications. Fig. 7 shows the graph between these two factors and shows that target 1 has a shorter response time.

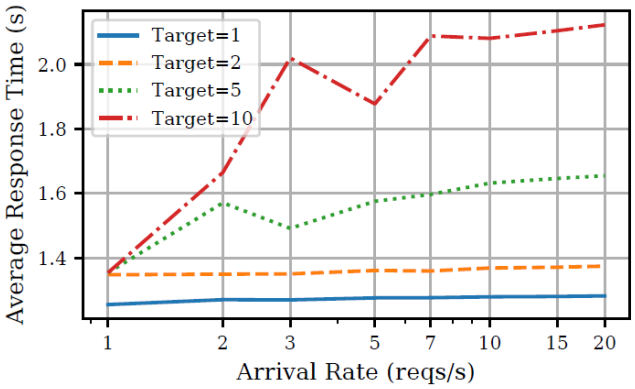

Fig. 8. Average Response time vs. Arrival time.

Figure 8 shows the average response time in the aspect of arrival rate in the x-axis. The graph shows the greater response time for a greater number of targets while target 2 shows the exceptional conditions.

The results of our study align with previous research by Ishakian et al. (2018), who also found that warm serverless function executions have reasonable latency, but cold starts incur considerable costs. This highlights the importance of considering bimodal latency distribution in serverless architectures, as failure to account for this may jeopardize adherence to SLAs (Service Level Agreements) [8].

Furthermore, our findings support the argument made by Castro et al. (2019) regarding the advantages of the Function-as-a-Service paradigm in simplifying operational complexity for developers. By abstracting away infrastructure management, serverless computing lowers the barrier for developers, enabling them to focus more on application logic.

The proposed Adaptive Function Placement (AFP) technique is in line with the work of Kaffes et al. (2020), who emphasized the importance of efficient resource allocation in serverless computing platforms. Our study extends this work by demonstrating how statistical machine learning can be used to optimize function placement dynamically, leading to improved performance and cost-efficiency.

V. Conclusion

In this paper, the author has suggested and assessed a performance model for serverless computing platforms' metric-based auto-scaling that is precise and manageable. It examines the effects of various system topologies and the workload characteristics of these systems and uses experimental validation to demonstrate the efficacy of the suggested model. It is also demonstrated how application owners can utilise the presented performance model as a tool to determine the best configuration for a particular workload under various loads. The suggested methodology can also be used by serverless providers to set adaptive defaults for the target value configuration that are more logical. In accordance with the real-time arrival rate, the performance of the system depends on the cost, energy, average response time, and energy consumed by the system. Monitoring and managing the cost of optimised resources and effective security mechanisms can be focused on in the future.

One of the key novelties of the research was the integration of machine learning models within containers to facilitate easy deployment and management in a serverless environment. By leveraging statistical machine learning techniques, the study showed how the AFP technique can improve the performance of serverless computing systems while reducing operating costs. Additionally, the research highlighted the importance of proper testing and validation of machine learning models to ensure reliable predictions and classifications on unseen data.

The study primarily focuses on Knative as the serverless computing platform for evaluation. While Knative is a widely used platform, its performance characteristics may not fully represent other serverless platforms. Future studies could explore multiple serverless platforms for a more comprehensive analysis.

The experiments were conducted using a simplified workload, which may not fully capture the complexity of real-world applications. Future work could involve more diverse and realistic workloads to better assess the proposed system's performance and scalability.

Suggestions for further study include exploring the scalability and efficiency of the AFP technique in larger and more complex serverless computing environments. Additionally, further research could investigate the integration of other advanced machine learning algorithms and techniques to enhance the performance and adaptability of serverless computing systems.

Acknowledgment

This work does not support by any financial organization.

References

[1] Vashisht, P., & Kumar, V. (2022). A Cost Effective and Energy Efficient Algorithm for Cloud Computing. International Journal of Mathematical, Engineering and Management Sciences, 7(5), 681-696. https://doi.org/10.33889/IJMEMS.2022.7.5.045.

[2] Anand, A., Das, S., Singh, O., & Kumar, V. (2022). Testing resource allocation for software with multiple versions. International Journal of Applied Management Science, 14(1), 23-37. 5.

[3] Apte, V., Viswanath, T. V. S., Gawali, D., Kommireddy, A., & Gupta, A. (2017). AutoPerf. In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering. ICPE '17: ACM/SPEC International Conference on Performance Engineering. ACM. https://doi.org/10.1145/3030207.3030222.

[4] Bardsley D., L. Ryan, and J. Howard, "Serverless Performance and Optimization Strategies," in 2018 IEEE International Conference on Smart Cloud (SmartCloud), IEEE, 2018, pp. 19–26.

[5] Bortolini D. and Obelheiro R. R., "Investigating Performance and Cost in Function-as-a-Service Platforms," in International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Springer, 2019, pp. 174–185.

[6] Castro, P., Ishakian, V., Muthusamy, V., & Slominski, A. (2019). The rise of serverless computing. In Communications of the ACM (Vol. 62, Issue 12, pp. 44–54). Association for Computing Machinery (ACM). https://doi.org/10.1145/3368454.

[7] Hellerstein J. M. et al., "Serverless computing: One step forward, two steps back," arXiv preprint arXiv:1812.03651, 2018.

[8] Ishakian, V., Muthusamy, V., & Slominski, A. (2018). Serving Deep Learning Models in a Serverless Platform. In 2018 IEEE International Conference on Cloud Engineering (IC2E). 2018 IEEE International Conference on Cloud Engineering (IC2E). IEEE. https://doi.org/10.1109/ic2e.2018.00052.

[9] Kaffes, K., Yadwadkar, N. J., & Kozyrakis, C. (2019). Centralized Core-granular Scheduling for Serverless Functions. In Proceedings of the ACM Symposium on Cloud Computing. SoCC '19: ACM Symposium on Cloud Computing. ACM. https://doi.org/10.1145/3357223.3362709.

[10] Li, X., Liu, S., Pan, L., Shi, Y., & Meng, X. (2018). Performance Analysis of Service Clouds Serving Composite Service Application Jobs. In 2018 IEEE International Conference on Web Services (ICWS). 2018 IEEE International Conference on Web Services (ICWS). IEEE. https://doi.org/10.1109/icws.2018.00036.

[11] Liu, B., Li, P., Lin, W., Shu, N., Li, Y., & Chang, V. (2018). A new container scheduling algorithm based on multi-objective optimization. In Soft Computing (Vol. 22, Issue 23, pp. 7741–7752). Springer Science and Business Media LLC. https://doi.org/10.1007/s00500-018-3403-7.

[12] Lloyd, W. J., Pallickara, S., David, O., Arabi, M., Wible, T., Ditty, J., & Rojas, K. (2017). Demystifying the Clouds: Harnessing Resource Utilization Models for Cost Effective Infrastructure Alternatives. In IEEE Transactions on Cloud Computing (Vol. 5, Issue 4, pp. 667–680). Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/tcc.2015.2430339.

[13] Lloyd W., Ramesh S., Chinthalapati S., L. Ly, and S. Pallickara, "Serverless Computing: An Investigation of Factors Influencing Microservice Performance," in 2018 IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2018, pp. 159–169.

[14] Ye, K., Wu, Z., Wang, C., Zhou, B. B., Si, W., Jiang, X., & Zomaya, A. Y. (2015). Profiling-Based Workload Consolidation and Migration in Virtualized Data Centers. In IEEE Transactions on Parallel and Distributed Systems (Vol. 26, Issue 3, pp. 878–890). Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/tpds.2014.2313335.

[15] Van Eyk E., A. Iosup, C. L. Abad, J. Grohmann, and S. Eismann, "A SPEC RG Cloud Group's Vision on the Performance Challenges of FaaS Cloud Architectures," in Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, ACM, 2018, pp. 21–24.

[16] Verma, S., Gupta, A., Kumar, S., Srivastava, V., & Tripathi, B. K. (2020). Resource allocation for efficient IOT application in fog computing. International Journal of Mathematical, Engineering and Management Sciences, 5(6), 1312.

[17] Wang L., M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking Behind the Curtains of Serverless Platforms," in 2018 USENIX Annual Technical Conference (USENIX ATC 18), 2018, pp. 133–146.

[18] Yin, F., Dong, D., Lu, C., Zhang, T., Li, S., Guo, J., & Chow, K. (2018). Cloud-Scale Java Profiling at Alibaba. In Companion of the 2018 ACM/SPEC International Conference on Performance Engineering. ICPE '18: ACM/SPEC International Conference on Performance Engineering. ACM. https://doi.org/10.1145/3185768.3186295.