Queue-namics
Auto Scaling Through Queuing Theory*

* Project Report for Principles of Cloud Computing

1st Sparsh Amanrnani
Masters Student
Vanderbilt University
Nashville, USA
sparsh.r.amarnani@vanderbilt.edu

2stArpit Ojha
Masters Student
Vanderbilt University
Nashville, USA
arpit.ojha@vanderbilt.edu

***Abstract*—This project introduces a novel Autoscaling service designed to optimize compute resource management in enterprise applications with variable workloads. We present a Simulator for AutoScaling, developed using MATLAB Simevents, that allows for the rapid evaluation of autoscaling strategies without the need for cloud deployment. Our contributions include a MATLAB-based simulator, a queuing theory-based compute cluster model, and an implementation on Kubernetes. Experimental results demonstrate the potential of our approach, with future work aimed at refining the simulation accuracy and controller algorithm, enhancing workload forecasting, and extending comparative testing with other autoscalers. Our research advances the field of dynamic resource management in distributed systems.**

***Index Terms*—Cloud Computing ,AutoScaling ,MATLAB ,Kubernetes**

I. Introduction

In the dynamic landscape of enterprise applications, managing compute resources efficiently is a critical challenge. These applications often experience fluctuating workloads throughout the day, leading to periods of underutilization during low demand and potential breaches of Quality of Service (QoS) requirements during peak times. Addressing this challenge necessitates a robust solution capable of adapting to changing demands—an Autoscaling service. Such a service dynamically adjusts the number of compute resources in response to incoming workload, ensuring optimal resource utilization and consistent QoS.

However, the deployment of autoscaling resources, even in a testbed environment, is not without its complexities. It is a time-consuming endeavor that requires meticulous configuration and extensive data collection about the virtual machines (VMs) in use. Furthermore, comparing the performance of different autoscaling approaches is often complicated due to the variability in their configuration requirements and the data they necessitate.

To tackle these issues, this project aims to develop a comprehensive testing framework—a Simulator for AutoScaling. This simulator will facilitate the rapid evaluation of various autoscaling strategies, streamlining the comparison process and obviating the need for actual cloud deployment. By leveraging MATLAB Simevents, the simulator will provide a platform where different autoscaling approaches can be tested within model microservice-based architectures. This will enable us to analyze the impact of events such as workload increases, response time fluctuations, and efficiency changes with ease.

The contributions of this project to the existing body of knowledge on Autoscaling are threefold. First, we will build a simulator using MATLAB Simevents, which will serve as a testbed for different autoscaling strategies. Second, we will describe a model of a compute cluster based on Queuing theory, specifically an M/M/C queue, which will underpin our understanding of resource allocation dynamics. Lastly, we will implement our framework on Kubernetes to show the effectiveness of our approach.

Section 2 provides an overview of existing research, examining different methodologies for implementing autoscaling within distributed systems. In Section 3, we detail the architectural elements of our autoscaling solution, including its design in MATLAB and practical deployment using Kubernetes. Section 4 outlines the experiments conducted to evaluate our autoscaling method against two established baseline strategies. Finally, Section 5 summarizes our project's findings and outlines potential avenues for further exploration and enhancement of our work.

II. Literature Review

I read four papers in my literature review.

1) A Survey of Autoscaling in Kubernetes : The Paper surveys the state of the art of existing approaches to solve the problem of container autoscaling in Kubernetes, their main characteristics as well as their current issues. Kubernetes has a build in Autoscaler however it is a reactive autoscaler and people have tried coming up with other approaches. It discusses proactive autoscaling approaches which use machine learning, deep learning and reinforcement learning algorithms to achieve optimal resource allocation.
2) Elastic Cloud Services: Scaling Snowflake's Control Plane : The paper talks about Snowflake's Elastic Cloud

Services which provides a bunch of management and orchestration services for Snowflake Data Cloud, like autoscaling , software rollout and throttling. Their Dynamical Thortlling approach sets concurrency limits on each VM which are dynamically recalculated based on the VM's utilisation every 30 seconds, which if exceded request to the VM's are rejected untill more resources are free. They implement a horizontal autoscaler with that where the number of desired VMs are calculated as ratio of the current load over the desired load. It defines unhealthy behaviour and such that if certain memory pressure and CPU loads are exceded its Throttling and Autoscaling mechanism is triggered.

3) Efficient Autoscaling in the Cloud using Predictive Models for Workload Forecasting : This paper proposes Autoscaling as a resource optimization problem and develops a model predictive algorithm to deal with it. They first use an ARIMA model to forecast upcoming workload. It uses a Customer Behavior Modeling Graph which is a Bayes net telling us the probability of customers visiting various pages in a graph, build from customer logs. The paper then proposes a unique receding horizon control algorithm to solve an optimisation problem of choosing the best action based on its SLA violation, cost of leasing a machine per hour and cost of reconfiguring a machine.
4) Power and performance management of virtualized computing environments via lookahead control : This paper implements a resource provisioning framework for cloud environments and poses autosaling as a utility maximization problem under risk. They also use trace based simulation to analyses how their approach performs on bigger computing environments. They solve their optimization problem using Limited Lookahead Control and it also proposes two controllers which solve different parts of the problem. It uses Kalman Filters to predict upcoming workloads, but it adds an error rate/ confidence bound on each prediction which gives it three possible arrival rates for upcoming workloads. It explicitly models how their clusters response time and queue length evolves over time using sytem dynamics equations. And their autoscaler is not only allowed to allocate more VMs but change VM configurations, turn of host machines and load balance amongst cluster. They make this decision considering the SLA violations created, power consumed and switching cost.

The 3rd and 4th paper have been the most relevant to our work and we've taken inspiration of forecasting workload using ARIMA models from the 3rd and inspiration for our objective function, system dynamics from 4th.

III. Methodology

A. Application Architecture

we'll define an application architecture that we will be working with and that we will use for our auto scaler as testbed.

It consists of 4 host machines. The first one represents a user .The second one a data streaming service like Apache Kafka which can make data continuously available in a given channel. Our third machine is a cluster of VMs which all run some machine learning inference model trying to predict a target variable from the data given by the first machine. We also have a cluster of databases which store the data sent by the user and the prediction results sent by the Machine Learning inference model. Our controller obtained an arrival list from our IOT/ Producer model which tells how many request arrived every 20 seconds and it uses this list to forecast the arrivals for the next 200 seconds. We will need to analyse the effect of our arrivals on our cluster and we do that through a queue simulator. Our controller interacts with our queuing simulator testing the effects of various servers and serving time on our queue for a given workload prediction and gives out an optimal prediction using a novel algorithm created by me called "Heuristic Single Step Look Ahead Search" which is a simplified Limited Look Ahead Search for a single time horizon and uses a Heuristic Search.

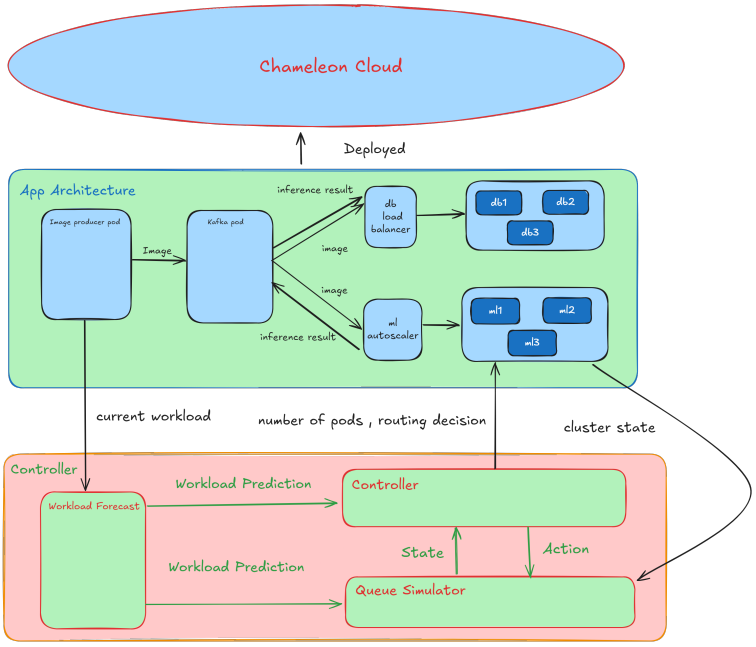

Fig. 1. Application Architecture

B. Modeling Clusters as a Queue

A M/M/C stochastic process where a bunch of customers arrive at a Arrival rate M and served by C servers in parallel which have a serving time with a serving rate of M.

To model our application architecture we will treat our data streaming service as M/M/2 queue where its two servers the ML and Database cluster which are also M/M/C queues.

M/M/C queues are useful as with a given distribution of Arrival rate and Serving rate we can predict many properties

about the queues, like average response time, average queue length, queuing intensity and more. We can simulate our queues as well to get the exact exit times of customers.

C. Problem Definition

With our Queuing model and Application architecture specified we can define our problem.

1) Let the state of a queue be given by

$$x_i(k) = (r_i(k), q_i(k), n_i(k))$$

Where r represents the average response time of the cluster and q represents the length of the queue

2) Let the control input of our auto scaler be defined as

$$u_i(k) = (n_i(k), f_{ij}(k)$$

We consider the the number of servers (n) and the server efficiency (f) allocated, which comes out as serving time.

3) Let the our arrival rate be given as

$$\lambda_i(k) \cong \lambda_i^*(k)$$

Customer arrival rate is our environmental input, but we can't know the exact input so we estimate it (we are considering ARIMA)

4) Let our transition function be

$$x_i(k+1) = \phi(x_i(k), u_i(k), \lambda_i(k))$$

Where phi is our simulation model, which can give us the average response time and queue length given customer arrivals and controller input, and previous state for a given time stepOur transition function is given by this algorithm :
It first queues all the arriving entities until a server gets free. After which it simulates the server serving the entity until all servers are occupied. Each entity response time is calculated as the sum of its waiting time and its time being served. And the algorithm returns queue length and average response time in the end.

Algorithm 1 M/M/C Queue Simulation

function MMC($arrivals$, $queue$, $serving\_time$, $time\_horizon$, $servers$, $time\_delta$)
  $resp\_time \leftarrow [0]$
  $j \leftarrow 0$
  while $j < time\_horizon$ **do**
    $arr \leftarrow arrivals[j]$
    for $i \leftarrow 1$ **to** $\min(servers) + 1$ **do**
      if $arr > 0$ **and** $j < time\_horizon$ **then**
        $init\_job \leftarrow [serving\_time, 0]$
        $jobs\_array \leftarrow [init\_job] * arr$
        $jobs \leftarrow jobs\_array$
        $queue \leftarrow$ STACK($queue$, $jobs$)
      end if
      if $queue$ **is not empty then**
        $time\_added\_per \leftarrow [0, time\_delta]$
        $time\_added \leftarrow [time\_added\_per] *$ LENGTH($queue$)
        $queue \leftarrow queue + time\_added$
      end if
      $j \leftarrow j + 1$
    end for
    $s\_next \leftarrow$ ARGMIN($servers$)
    $servers \leftarrow servers - servers[s\_next]$
    while $\min(servers) == 0$ **and** $queue$ **is not empty do**
      $top \leftarrow queue[0]$
      $servers[s\_next] \leftarrow queue[0, 0]$
      $resp\_time \leftarrow$ APPEND($resp\_time$, $\sum(top)$)
      $queue \leftarrow$ DELETE($queue$, 0)
      $s\_next \leftarrow$ ARGMIN($servers$)
    end while
    for all $s \in servers$ **do**
      $s \leftarrow \max(s - 1, 0)$
    end for
  end while
  $avg\_response\_time \leftarrow$ AVERAGE($resp\_time$)
  $queue\_length \leftarrow$ LENGTH($queue$)
  return $avg\_response\_time$, $queue\_length$
end function=0

5) We will now define our objective function :
   a) We define our relation between revenue and response time of a cluster, if the response time goes before a certain threshold then we go in a loss

$$H_i(r_i(k))$$

   b) We define our relation between power consumption and VMs hosted on a machine

$$S(n_i(k))$$

$$P(n_i(k+1), n_i(k)) = 0$$

   c) Profit Generated at time k by taking an action is given by

$$R(x(k), u(k)) = \sum_{i=1}^{2} (H_i(r_i(k+1)) - E * S(n_i(k)))$$

6) Thus our optimisation problem is defined over a given time horizon h

$$\max_u \sum_{l=k+1}^{k+h} R(.)$$

Subject to :

$$5 >= n_i(l) >= 1$$

We want to choose an optimal value action at time k which will maximise our expected reward over the given time horizon h.

Our first constraint set a limit to minimum number of VMs running on our cluster as well as a maximum limit which would be related to the computation power of the host machine.

D. Controller

Our controller interacts with our Queue Simulator and uses an algorithm which we call the "Heuristic Single Step Look Ahead Search" , its a simplification of Limited Look ahead Control algorithm and fixes a single action over a given time horizon and then searches for better action based on the performance of the previous action. Although it computes a fixed strategy for a single time horizon i has a more favorable time complexity and fixed time complexity for on decision relying only on the maximum number of servers moreover in our testing it has shown that to never violate an SLA Response Time.

E. MATLAB Implementation :

We tested our auto scaler in MATLAB using the Sim Events Toolbox. Sim Events is a discrete event simulator to model communication and queuing systems. The model is presented in Fig 4. It helped me develop and test my components easily and also test my controller against baselines.

F. Deploying on Cloud :

We deployed our application and controller on Chameleon Cloud. We used VGG11 and CouchDB as our Machine Learning Server and Database Respectively and use Kubernetes to orchestrate our application as pods. We deployed our server behind a load balancing service which allowed it to scale up effectively.Our workload creating was done using an Amplitude increasing cosine wave as the time between our workload.

IV. Experiments

A. MATLAB Setup :

we've set up our experiment in MATLAB using the SimEvents module, The Entity Generator represents incoming user workload, The Entity Servers represent , VMs in the cloud that might run application to serve a request. The queue represents the load balancer of the cluster. we've setup 1 queue representing the Kafka Broker and two clusters which both communicate with the broker.

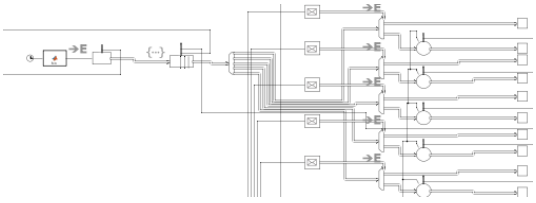

Fig. 2. Experimental Setup

Algorithm 2 Heuristic Single-Step Lookahead Search

function HEURISTIC-SINGLE-STEP-LOOKAHEAD-SEARCH($time\_horizon$, $arrivals$, $serving\_time$, sla_rt, $max\_servers$, $time\_delta$)
  $max\_c \leftarrow -1$
  $max\_reward \leftarrow -10000$
  $st \leftarrow serving\_time$
  $min\_st \leftarrow st$
  $min\_rt \leftarrow -1$
  $min\_ql \leftarrow -1$
  for $c \leftarrow 1$ **to** $max\_servers$ **do**
    $queue \leftarrow$ CREATEARRAY$(2, 2, \text{fill} = 0)$ {Equivalent to np.array([[0,0],[0,0]])}
    $servers \leftarrow$ CREATEZEROSARRAY(c) {Equivalent to np.zeros(c, dtype=int)}
    $avg\_response\_time, queue\_length \leftarrow$ MMC$(arrivals, queue, st, time_horizon, servers, time_delta)$
    $sla\_gain \leftarrow 0$
    $e\_consumption \leftarrow 0.5 \times c/max\_servers$
    if $avg\_response\_time \leq sla\_rt$ **then**
      $sla\_gain \leftarrow 1$
    else
      $sla\_gain \leftarrow -1$
      $st \leftarrow$ ROUND$(st \times (1 - c/max_servers))$
    end if
    $reward \leftarrow sla\_gain - e\_consumption$
    if $reward > max\_reward$ **then**
      $max\_c \leftarrow c$
      $max\_reward \leftarrow reward$
      $min\_rt \leftarrow avg\_response\_time$
      $min\_ql \leftarrow queue\_length$
      $min\_st \leftarrow st$
    end if
  end for
  $best\_c \leftarrow max\_c$
  $best\_st \leftarrow min\_st$
  return $best\_c, best\_st$
end function=0

B. ARIMA Forecast :

We use an ARIMA model to forecast upcoming workloads for up to 10 timesteps, wherewith these parameters 1,q=1,d=1. Our model can forecast its trend very well but cannot precisely predict the next workload.

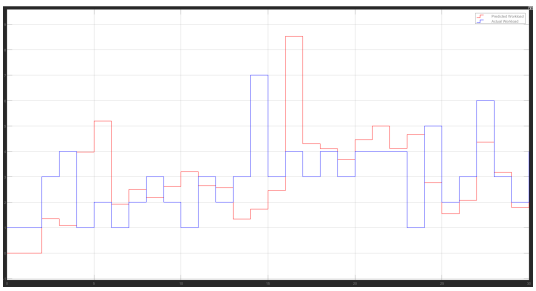

Fig. 3. ARIMA Forecast

Metrics that we should consider : We'll measure Average Response Time and , Percentage of SLA violations , Utilisation of each server.

C. Controller Output :

Our controller adjusts the our server time and servers based on if the previous output had an SLA violation , this ensures that an action is picked which doesn't have an SLA violation. Our output doesn't ensure that an optimal SLA violation is reached, infact it purposely chooses a suboptimal one to ensure that there aren't any SLA violation. Here is our output for the workload we generated.

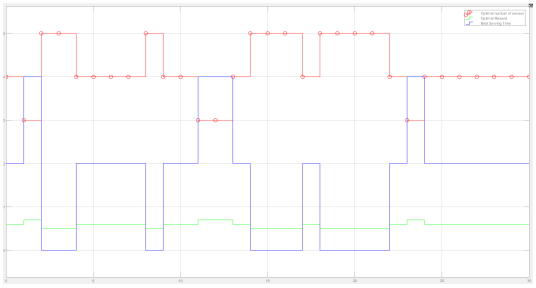

Fig. 4. Controller Output

Metrics that we should consider : We'll measure Average Response Time and , Percentage of SLA violations , Utilisation of each server.

D. Metrics for Evaluation Average Response Time and Reward Function :

We calculate the Average Response Time using the utilisation values of each server and the average wait of our queue.

$$R_c = \frac{\sum_{i=1}^{5} U_i}{5} * 30 + AvgW_{queue}$$

Our Response Time of our cluster is defined as the Utilisation of all the servers multiplied by our simulation time added to our average queue wait.

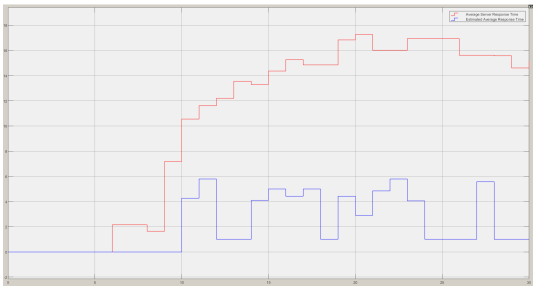

Fig. 5. Controller Response Time

- Baseline 1 : Chooses the first server with 8 serving time.

Fig. 6. Baseline1 Reward

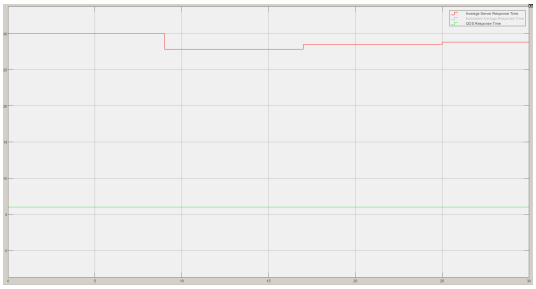

Fig. 7. Baseline1 Response Time

- Baseline 2 : Chooses all the 5 servers with 8 serving time.

Fig. 8. Baseline2 Reward

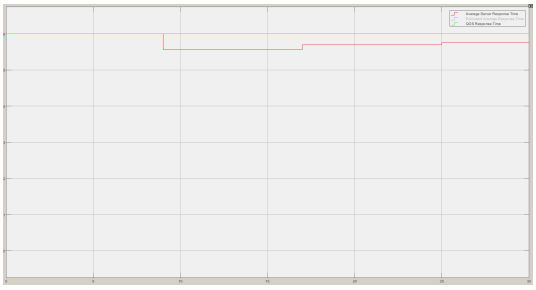

Fig. 9. Baseline2 Respose Time

- Controller : ”Heuristic Single Step Look Ahead Search” to decide its action.

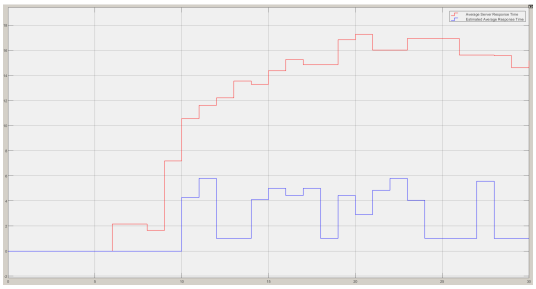

Fig. 10. Controller Response Time

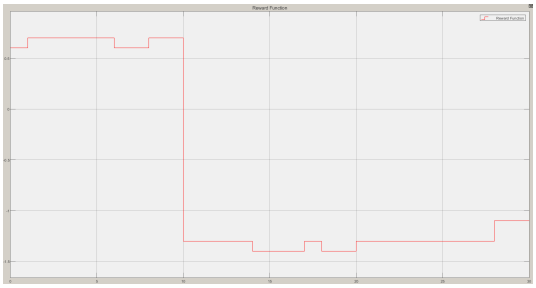

Fig. 11. Controller Reward

We can see that while our controller does adjust to upcoming workload it can use give rewards that are better than our baselines for at many instances, there are times where our baselines performs better especially baseline 2 which performs better after t=10.

E. Kubernetes Deployment

we deployed our application on Kubernetes on Chameleon Cloud. One of our limitations was translating the discrete simulation to a continuous one as translating the service time for our servers could be floating values and thus our queue simulation function limits the clarity we have over our system as it is being used in the for loop as an integer, here is the result from our deployment. We tested our controller response across three variations of workload. All three workloads followed an increasing positive sine wave pattern, where the time between each request decreased at the mean of the wave. We set our Service Level Agreement (SLA) response time to 14 for the first and third workloads, and to 12 for the second workload. While we started with 6 replicas for the third workload, we began with 0 replicas for the first two workloads. In our experiment, we measured the reward, response time, and the optimal number of replicas.

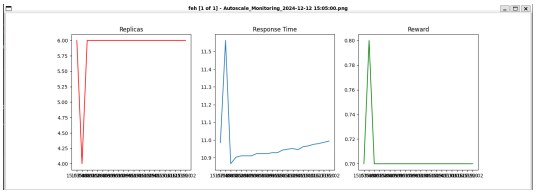

Fig. 12. First Workload

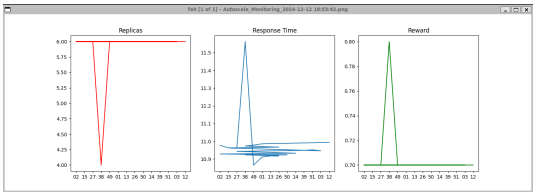

Fig. 13. Second Workload

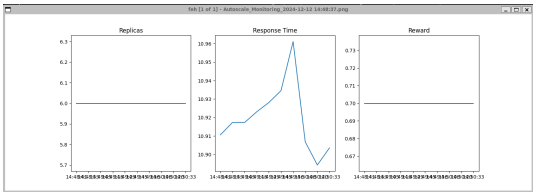

Fig. 14. Third Workload

V. Conclusion

My work can be found here : GitHub Link. our class project implemented an Autoscaler with a new novel simulation approach and a novel algorithm for our controller. More work needs to be done on our controller to make sure it doesn't get stuck in minimas and on our queue simulator to accurately predict the state of our cluster. we believe implementing state updates from our cluster would significantly improve our approach. Where we could check for the queue length and response time from our cluster and use that as the start of our simulation for our controller. Moreover better forecasting models can be used to predict upcoming workload, along with trying to adjust the hyper parameters of the ARIMA model. Moreover we can incorporate more comprehensive testing can be done, where we can test our approach against other autoscalers