A Prediction based Autoscaling in Serverless Computing

Ha-Duong Phung, Younghan Kim
School of Electronic Engineering, Soongsil University, Seoul, Korea

TCSS 562 Team 15:
Yuan Huang, Yifan Xie, Alan Liu

UNIVERSITY of WASHINGTON

1

OUTLINE

- Introduction
- Describe the Knative and analyze some related forecasting models.
- Detail of the proposed method structure.
- Presents the results showing how different concurrency limits impact performance and the efficiency of their prediction model Bi-LSTM compared to the Knative default.
- Conclusion and the future work plans.

UNIVERSITY of WASHINGTON

2

Introduction: Paper Overview

> **What is Knative?**

> **What is Auto-Scaling?**

Problem being solved:
Knative cannot adapt well to the fluctuation of a request's trend.

3

Introduction: Paper Overview

Why is it a problem?

1. Users have no idea about the appropriate parameter values, while those predefined values can strongly impact the performance.
2. The negative influence can cause response time increase and throughput reduction if those are overprovisioning or under-provisioning.
3. Knative uses a moving average method to calculate the number of pods based on past data but cannot reflect the future workload trend, leading to the delay effect.

Why is it a problem the research community is interested?

Due to the fast growth of serverless computing offerings, they focus on maximizing performance with the lowest resource utilization and optimizing response time to satisfy the quality of service (QoS) requirements so that it is content with users' experience while users only must pay the least cost.

4

Introduction: Paper Overview

Address the problem:

Automatically optimizes an effective scaling policy to a specific application.

How?

- With the same amount of resources per pod, the purpose focuses on outputting an appropriate parameter level showing the throughput maximum and an acceptable latency to satisfy SLO violation.
- A new service revision with the new scheduling policy is created, and then apply the forecasting model de to optimize the calculation of the number of pods.

5

BACKGROUND

A. Knative Serverless Platform

B. Forecasting model

6

A. Knative Serverless Platform

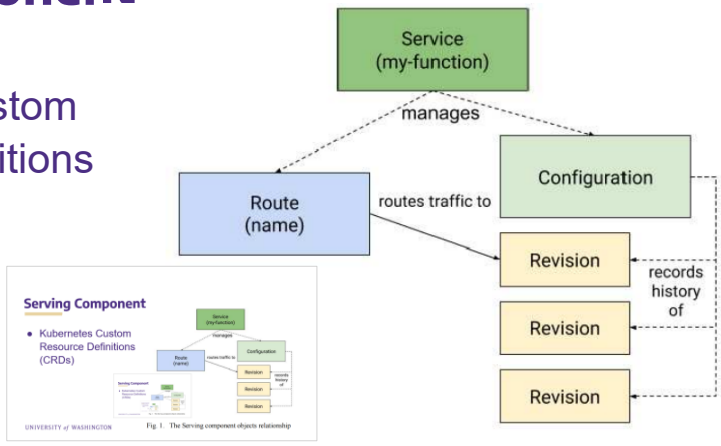
- open source serverless platform
- two main components:
  - Eventing component
  - **Serving component**

UNIVERSITY of WASHINGTON

7

Serving Component

- Kubernetes Custom Resource Definitions (CRDs)



UNIVERSITY of WASHINGTON

Fig. 1. The Serving component objects relationship

8

Auto-scaling

- Knative Pod Autoscaler (KPA)

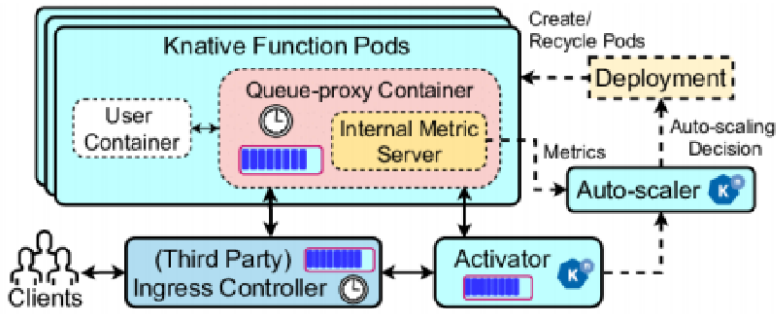


Fig. 2. The Serving component objects relationship

Auto-scaling metrics

- Concurrency
- request per second (rps)

$$N = \frac{c_t + c_{t-1} + \cdots + c_{t-k+1}}{C_p * k}$$

Moving average method to compute number of pods required in the future

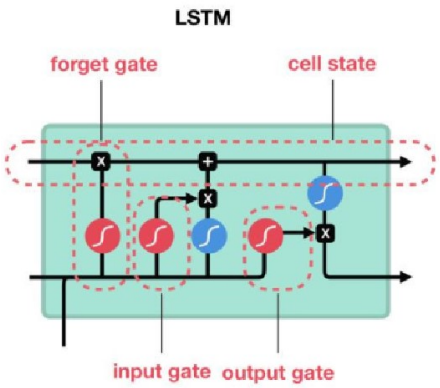
N: Number of pods required
Cp: Maximum request concurrency one pod can handle at any time
Ct: Actual average concurrency per pod at time t

Drawbacks

- not reflect trend
- how to choose Cp

B. Forecasting model

- **machine learning algorithm** to **time series forecasting**
- traditional method - not applicable
- **Deep neural method** - applicable
- **LSTM** and **Bi-LSTM**



Proposed Architecture

- Two primary states to address above challenges.
  - First State: Parameter values for the primary service function to optimize performance & the latency of service
  - Second State: Improve the calculation of the number of pods to be more adaptive to the changes in request trends by applying the prediction model Bidirectional LSTM(Bi-LSTM)
- The activator controls these two states, which run separately

UNIVERSITY of WASHINGTON

13

Proposed Architecture

- Finding appropriate configurations for parameters of the service version and using them to create a new service revision
- Knative automatically transfers all 100% current workload to the latest revision, and the activator redirects the system to state 2
- The time series forecasting model Bi-LSTM will be applied to improve the calculation of the number of pods. The decision controller controls the flow and executes results received from two states by communicating with Knative components

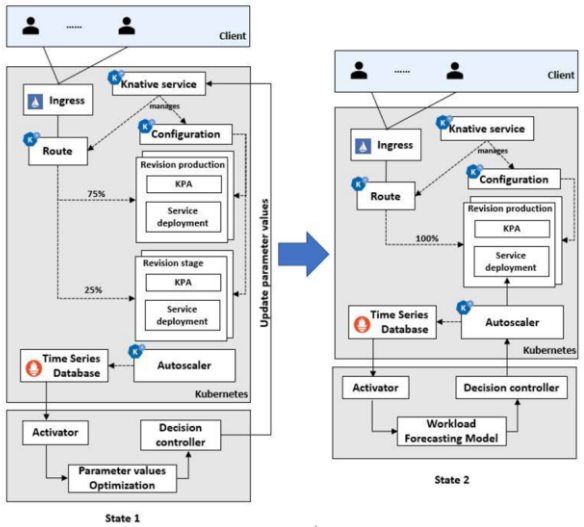


Figure 3. System architecture of proposed method

UNIVERSITY of WASHINGTON

14

Proposed Architecture

- Knative resources installed on Kubernetes with the help of service mesh Istio
- All incoming requests will pass through the Istio ingress before coming to Knative serving components, which control the state of the deployed sample service and enable autoscaling of additional pods based on demand

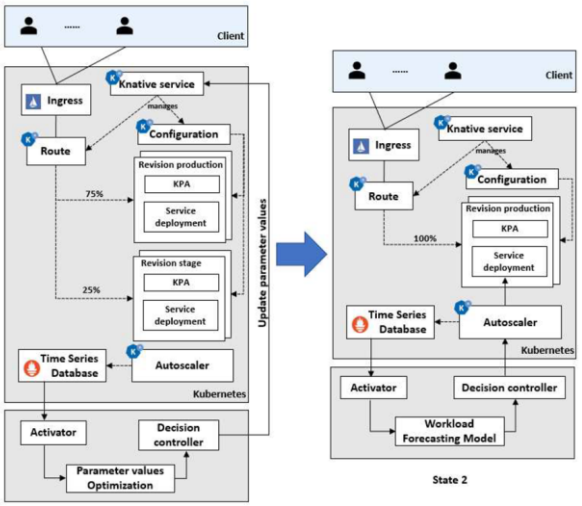


Figure 3. System architecture of proposed method

15

Proposed Architecture

- Split the workload into 2 kinds of revisions due to Knative serving flexibility: a stage revision and a production revision
  - The stage revision: testing & calculating the appropriate parameter values for the current load
  - The production revision: serve the requests continuously
- Route: distribute inbound requests between the stage and production revision on a 25-75% basis.
- Choose the value of 25% because results extracted from lower values of traffic (e.g., 5-10%) could be affected by the noisy behavior of the system

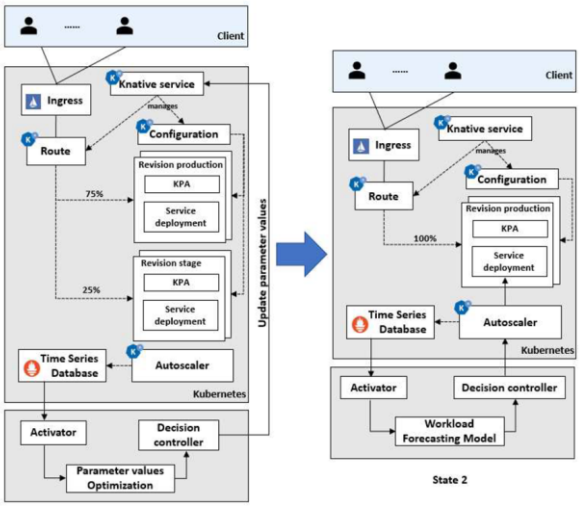


Figure 3. System architecture of proposed method

16

Proposed Architecture

Parameter Values Optimization

- Run performance tests for varying concurrency levels
- Collect the output information:
  - maximum throughput, request latency distribution, and corresponding concurrency level for each test iteration
- Begin at a concurrency limit of 10 and use steps of 10 to update the next limit. Stop when latency is over SLO violation
- Create a new service revision using the output and transfer 100% workload to the latest revision

17

Proposed Architecture

Workload Forecasting

- Extract metrics from Prometheus (the value of the current workload each time interval)
- Sum it up and have the dataset for training the model (80% of total for training and 20% for testing)
- Used ten historic workload values to predict the future workload in Bi-LSTM model
- Optimize the model by tuning params so as to make RMSE and MAE small

18

Preliminary Experimental Result

- Compare the proposed method to the Knative default configuration of the 100-concurrency level through two different applications
- Two applications with different resource consumption per request:
  - Application A: consumes CPU and memory exclusively
  - Application B: low resource usage
- An SLO violation is considered to maintain user experience (configured 250ms)

Preliminary Experimental Result

- The appropriate concurrency level of application A is 50 with 50 rps
- Compared with the 100-concurrency level, the result not only is the latency low and acceptable but also the throughput is higher by approximately 40%

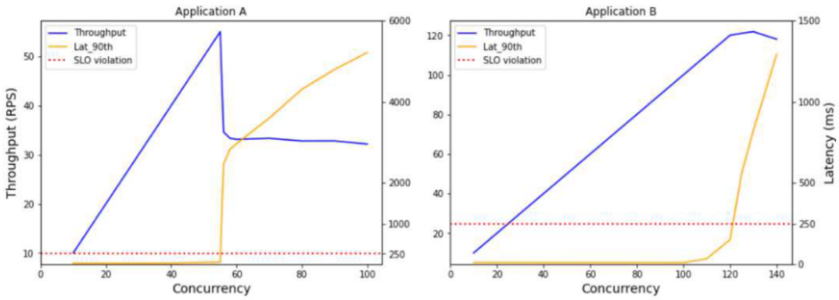


Figure 4. System architecture of proposed method

Preliminary Experimental Result

- The appropriate concurrency level of application B is 120 with 120 rps
- Compared with the 100-concurrency level, although the latency values of the two levels are acceptable, max throughput of our method is higher than 20%

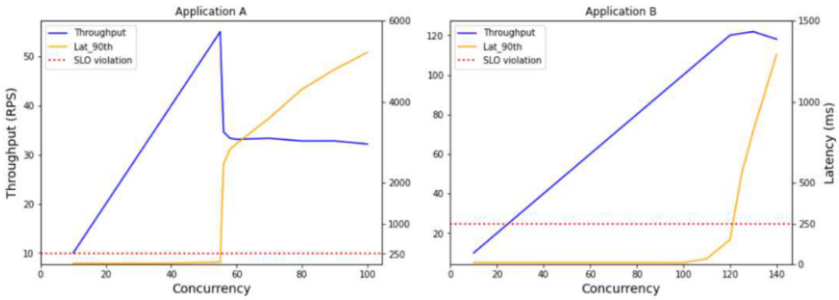
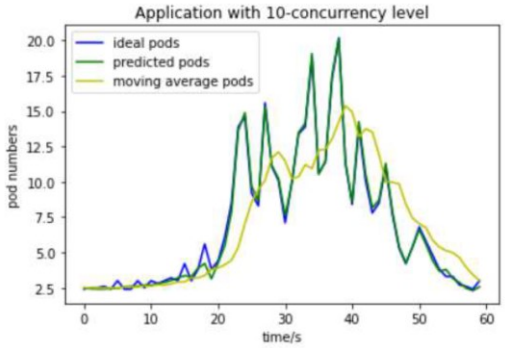


Figure 4. System architecture of proposed method

21

Preliminary Experimental Result

- The number of pods obtained based on the Bi-LSTM model is closer to the number of ho
- The predicted workload from the Bi-LSTM model is more accurate than the moving average, so the predicted number of pods value can be quickly adaptive to the workload trend



UNIVERSITY of WASHINGTON

22

Conclusion

- Find the appropriate parameter values for Knative service revision to improve performance and keep acceptable latency
- Optimize the calculation of the number of pods for Knative based on the prediction model Bi-LSTM
- The preliminary experiments worked well and show better performance than Knative scheme

Strengths

- Clear logic in paper design
- The underlying principle and implementation logic of the algorithm and forecasting models are introduced in detail
- The algorithm has a substantial improvement (over 20%) in performance and the effect is shown in the graphs
- Well-designed and described experimental architecture and controls

Weaknesses

- The amount of test cases could be small
- Few experiment details

Evaluation

- No result chart for the Knative default configuration of the 100-concurrency level for the two applications
- Result for workload forecasting is comparatively convincing

Gap

- Future works: plan to implement Knative's Autoscaler based on our experiments based on the production environment.

Thank you for watching!