2020 IEEE 9th International Conference on Cloud Networking (CloudNet) | 978-1-7281-9486-8/20/$31.00 ©2020 IEEE | DOI: 10.1109/CLOUDNET51028.2020.9335805

Supporting IoT Applications with Serverless Edge Clouds

I. Wang, E. Liri and K. K. Ramakrishnan
University of California, Riverside, CA
(iwang024@ucr.edu, eliri001@ucr.edu, kk@cs.ucr.edu)

***Abstract*—Cloud computing has grown because of lowered costs due to economies of scale and multiplexing. Serverless computing exploits multiplexing in cloud computing however, for low latency required by IoT applications, the cloud should be moved nearer to the IoT device and the cold start problem should be addressed. Using a real-world dataset, we showed through implementation in an open-source cloud environment based on Knative that a serverless approach to manage IoT traffic is feasible, uses less resources than a serverfull approach and traffic prediction with prefetching can mitigate the cold start delay penalty. However applying the Knative framework directly to IoT traffic without considering the execution context gives unnecessary overhead.**

I. Introduction

Advantages of IoT include universal connectivity and cheap sensors that reduce the high costs of manual labour. Actuators can respond to predefined sensor events while low power IoT devices support data gathering and may be used in multiple application scenarios. Serverless computing exploits multiplexing in cloud computing, however for low latency, the cloud should be moved nearer to the IoT device and the cold start problem should be addressed [1]. Edge clouds move the cloud closer to the device and fast, lightweight inference helps serverless computing anticipate IoT requests by learning from past IoT traffic patterns. This can help improve function provisioning, thus avoiding the latency penalty of cold start.

This work we discusses the key serverless components needed to support IoT traffic in the edge cloud and presents several IoT application scenarios to illustrate how microservices meet the needs of IoT traffic. With data from a real-world dataset, we then study the impact of this IoT traffic in an open-source cloud environment focusing on how the number and type of microservices affect performance. Our results show that a serverless approach uses less resources than the traditional cloud computing approach and prediction with prefetching reduces cold start delay penalty. However, the Knative design introduces significant overhead from a sidecar container called the queue-proxy, used for precise metrics measurement and probing for pod readiness. To fully benefit from a serverless implementation, it is important to understand and implement the appropriate components for the IoT context.

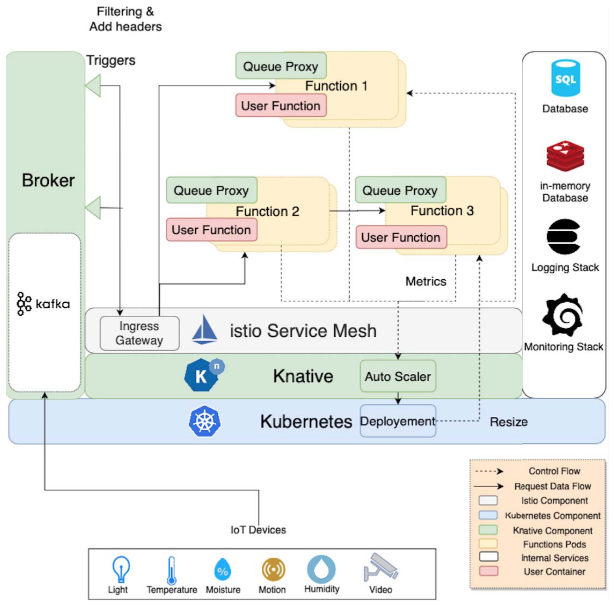

Fig. 1. Serverless architecture for IoT Service

II. Serverless for IoT

This section discusses the serverless components needed to support IoT traffic and some IoT application scenarios where serverless is a potential solution.

A. Serverless Computing

In serverless computing, the required components are dynamically instantiated to allow code execution when an event occurs. Dynamically adjusting the number of instances to meet demand is suitable for IoT traffic which can be highly variable.

Knative[2] is an open source serverless platform for dynamic container management built on top of Kubernetes, a container orchestration framework that automates the deployment and management of containerized workloads (see Fig. 1). Based on the observed Request per second(RPS) within an operator specified window, the autoscaler decides whether to scale up or down even to zero in order to meet current requirements, using performance metrics retrieved from Kubernetes or Knative sidecar containers such as the ”queue-proxy”. If a change in number of instances is required, Knative issues commands to the master node in the Kubernetes cluster to make the required adjustments.

B. Application Scenarios

We discuss two IoT applications where serverless is useful.

1) Home monitoring: In home monitoring, we consider entry/exit detection and environment monitoring. In this scenario, for simplicity, we consider detecting entry/exit into/from specific rooms in an apartment. We assumed each room has a single entrance and single occupant either entering or exiting the room. The nearest motion detectors to the door i.e. internal sensors inside the room and external sensors outside the room, send out events whenever a change of status occurs and these are used to detect entry/exit.

For environment monitoring we consider maintaining a preset room temperature. Temperature sensors periodically send updates to a function which can trigger a start/stop of the HVAC unit via an actuator.

2) Agriculture: In the agriculture scenario for simplicity we consider multiple temperature sensors regularly and periodically transmitting data to the cloud which is similar to how many agriculture IoT devices operate.

C. Serverless Edge Cloud

This section discusses setup of the edge cloud to support the experiments using CloudLab[3], Docker[4], Kubernetes and Knative. $Sensor functions$ receive data directly from sensors, $actuator functions$ send data to actuators or end users to trigger some action and request and response messages followed the CloudEvent specification. The entry/exit detection function chain has 2 layers (see Fig. 2). Internal motion detectors going on (i.e. entry event detected) triggers the chain to turn on the light. An exit event is detected if the following pattern occurs: internal sensor goes on, external sensor goes on and then the internal sensor goes off.When a motion sensor is triggered, the updated sensor status is written to Redis and the status of the door, lights and motion detector immediately on the other side of the door is checked. If an action is required, a message is sent to the actuator function to turn on/off the light. In the environment monitoring scenario, sensor readings parsed by the sensor function are sent to an actuator function to toggle the thermostat where necessary (see Fig. 2). Similar function chain design and logic applies to the agriculture scenario.

III. Experiment Design and Setup

We categorized sensors as High-Frequency Aperiodic (HFA) (motion), Low-Frequency Aperiodic (LFA) (temperature) and Periodic (agriculture) based on their reporting frequency. We used the CASAS dataset[5] to perform 10, 30 and 50 household experiments and examined cold start time and compared actual and predicted resource usage. Data for a single day(24 hours) in the dataset represented data from 1 household.

Linear regression using a single layer perceptron was used to predict the arrival time of the next packet from each sensor thus we can proactively start the sensor function before the traffic actually arrives. A customized autoscaler scaled the number of pods based on the predicted or actual total RPS. If no prediction is available or the difference between the predicted and actual RPS is larger than a threshold, the default policy is used. Response and CPU times measured responsiveness and performance while instant pod count and pod-minutes measured model accuracy and time averaged number of pods used.

TABLE I
Serverfull and serverless CPU usage for 1 household

Experiment	CPU time (s)
Serverless Cumulative CPU time	1.422747
Serverless:cumulative user container CPU time	0.012039
Serverless:cumulative queue proxy CPU time	1.410708
Serverfull (cumulative CPU time)	0.039711

IV. Results and Analysis

Fig. 3 compares the predicted and actual packet arrival intervals for periodic and aperiodic temperature data. The initial large difference between actual and predicted values for periodic data, causes under-provisioning of functions. One solution is to switch to the prediction mechanism only upon reaching a given prediction accuracy.

In the 10-household experiments, the temperature RPS is low, varies little (1-4 RPS) and needs only 1 pod while the motion RPS is generally higher, highly variable (0.3-171 RPS) and needs 1-4 pods since max RPS per pod is 50 (see Fig. 4 and 5). The serverfull (no multiplexing) case has a dedicated function pod per household while in the serverfull (with multiplexing) all households share function pods but provisioning is done based on the estimated peak pod requirement i.e., 1 pod for temperature and 4 pods for the motion sensors so most times there is over-provisioning.

For the 10 household motion experiment, measurements were taken every 15s and the pod min values for serverfull (no multiplexing), serverfull (with multiplexing) and serverless were 165, 66 and 42.75 respectively. Both serverfull cases are wasteful due to over-provisioning however the serverless solution uses less resources (1-4 pods) and approximately 64.8% of the resources used in the serverfull with multiplexing case. The serverless solution with dynamic autoscaling more closely follows the load requirements while meeting the SLA and confirming that a variable traffic pattern benefits from autoscaling. Larger scale aperiodic traffic experiments i.e., 30 and 50 household experiments also confirm this trend.

In the 10 household temperature experiment the pod min values for serverfull (no multiplexing), serverfull (with multiplexing) and serverless were 162.5, 16.25 and 16.25 respectively. To determine why serverfull (with multiplexing) and serverless have the same value we compared the CPU time which is the fraction of CPU used for a given time, as measured by Prometheus[6]. In a new experiment using 5 temperature sensors, we varied the frequency of sensor transmissions with a 5 minute maximum interval between transmissions allowing the function to idle periodically. The experiments ran for 720 mins and while the serverfull sensor function was active for the entire experiment, the serverless function was active only for 166 mins due to zero scaling.

Results show the serverless CPU time is much higher than the serverfull i.e. 1.422747s vs 0.039177s (see Table I). This is due to the serverless queue proxy sidecar container which runs when the user function pod is active. Considering

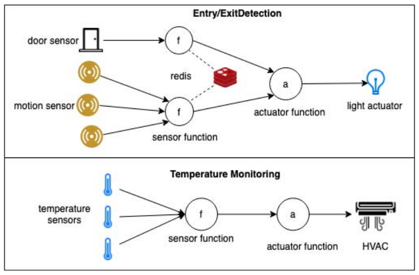

Fig. 2. Function chains for experiments

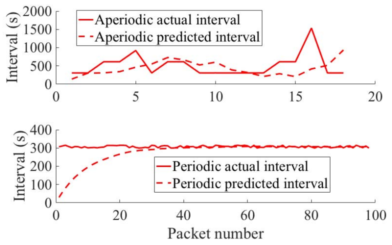

Fig. 3. Prediction (periodic & aperiodic data)

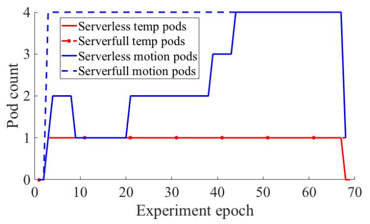

Fig. 4. Pod count for 10 household experiment

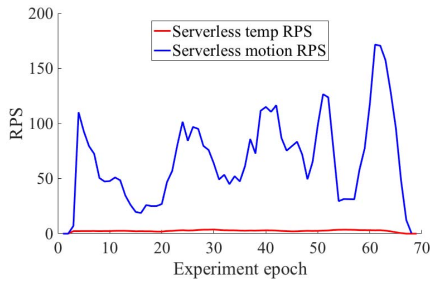

Fig. 5. RPS for 10 household experiment

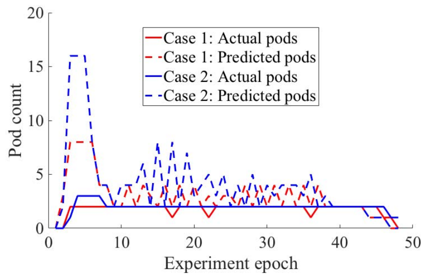

Fig. 6. Comparing predicted and actual pods used for periodic data

only user container CPU time, the serverless case uses 30% of the total CPU time compared to the serverfull case i.e. 0.012039s vs 0.039711s, which is reasonable due to serverless zero scaling. Instead of using the node Kubelet to probe readiness and collect function pod metrics, Knative instantiates separate queue-proxies for each function pod. This increases the overhead so for low IoT traffic it is highly desirable not to use a queue-proxy to avoid wasting CPU time. With higher IoT traffic that needs more function pods, instead of a queue-proxy, a more streamlined method using less CPU resources may be needed.

From the agriculture experiment we consider the impact of prediction and function prefetching with synthetic periodic data generated from 10 sensors with an interval of $300 \pm 5s$ in Case 1 and $300 \pm 15s$ in Case 2. In Fig. 6 the predicted is higher than the actual pod count so function prefetching (using the prediction) ensures more pods than required are available and so incoming requests do not experience a cold start.

Compared with case 1, case 2 has a larger variance in inter-arrival times, so the function is active longer and only 105 requests experience code start (i.e. response time >1s) (see Table II. More than a 10x reduction in packet count is seen in both cases when prediction and function prefetching is used.

TABLE II
CASES 1 AND 2 COLD START PACKET COUNT WITH/WITHOUT PREDICTION

Experiment	Cold start	No cold start	Average response time
Case 1: without prediction	1771	229	14.7356
Case 1: with prediction	10	1990	0.0309203
Case 2: without prediction	105	1895	0.763625
Case 2: with prediction	10	1990	0.0313907

V. Related Work

The performance of microservices varies up to 15x depending on the serverless infrastructure state[7]. The performance of open source serverless platforms is evaluated in [8] considering response time and ratio of successfully received responses to understand the design choices. [9] proposes a home/office monitoring system using IoT devices and multiple classifiers to determine when a home intrusion occurs. An activity aware approach was used to enhance security and detect threats in smart homes using machine learning and data from the CASAS home monitoring framework([10], [11], [5]).

Serverless is suitable for IoT however understanding traffic patterns is key for resource management and [12] compared the performance of four models when forecasting the traffic generation patterns of individual IoT devices.

VI. Conclusion

Serverless computing with autoscaling for IoT traffic allows better management of resources compared to a serverfull approach. We demonstrated this using serverless function chains based on Knative for home monitoring and agricultural applications. Zero scaling however introduces latency due to cold start but simple prediction and function prefetching can mitigate this for both low and high frequency request workloads. Despite the potential benefits with serverless computing, the Knative design introduces significant overhead from the queue-proxy sidecar container. We suggest that the queue-proxy may not be necessary for some IoT applications and believe the principles and results of our work are applicable to a broader class of IoT applications. Future work includes determining when a queue proxy is required and running a Knative solution without or with a light-weight queue proxy.

VII. Acknowledgment

We thank all the anonymous reviewers for their valuable feedback and the US NSF for their generous support of this work through grant CNS 1619441 and CNS-1763929.

References

[1] M. Shahrad, R. Fonseca, Í. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," *arXiv preprint arXiv:2003.03423*, 2020.

[2] Knative, "Knative." [Online]. Available: https://knative.dev

[3] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The design and operation of cloudlab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, Jul 2019, pp. 1–14. [Online]. Available: https://www.flux.utah.edu/paper/duplyakin-atc19

[4] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[5] W. S. University, "Center of advanced studies in adaptive system." [Online]. Available: http://casas.wsu.edu/datasets/

[6] Siebenmann, Chris, "Getting a cpu utilization breakdown in prometheus's query language, promql." [Online]. Available: https://utcc.utoronto.ca/ cks/space/blog/sysadmin/PrometheusCPUStats

[7] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless computing: An investigation of factors influencing microservice performance," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, 2018, pp. 159–169.

[8] S. Mohanty, G. Premsankar, and M. diFrancesco, "An evaluation of open source serverless computing frameworks," in *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2018, p. 115–120.

[9] A. Dhakal and K. K. Ramakrishnan, "Machine learning at the network edge for automated home intrusion monitoring," in *Proc. of Workshop on Machine Learning and Artificial Intelligence in Computer Networks, 25th IEEE International Conference on Network Protocols (ICNP)*, 2017, pp. 1–6.

[10] J. Dahmen, B. L. Thomasand D. J. Cook, and X. Wang, "Activity learning as a foundation for security monitoring in smart homes," in *Sensors*, vol. 17, no. 4, 2017, p. 737. [Online]. Available: https://doi.org/10.3390/s17040737

[11] D. Cook and M. Schmitter-Edgecombe, "Assessing the quality of activities in a smart environment," in *Methods of Information in Medicine*, 2009.

[12] M. Nakip, B. C. Gül, V. Rodoplu, and C. Güzeliş, "Comparative study of forecasting schemes for iot device traffic in machine-to-machine communication," in *Proceedings of the 2019 4th International Conference on Cloud Computing and Internet of Things*, ser. CCIOT 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 102–109. [Online]. Available: https://doi.org/10.1145/3361821.3361833