Power to the Applications: The Vision of Continuous Decentralized Autoscaling

This is the author version of the work. Original publication available soon in IEEE Xplore
Martin Straesser, Stefan Geißler, Tobias Hoßfeld, Samuel Kounev
University of Würzburg, Würzburg, Germany
Email: {martin.straesser, stefan.geissler, tobias.hossfeld, samuel.kounev}@uni-wuerzburg.de

***Abstract*—Autoscaling has been one of the most active research areas since the beginning of the cloud computing era. Nearly all previously proposed approaches focus on decision-making based on averaged monitoring values of many service instances at fixed points in time. This limits responsiveness and can lead to service level objective (SLO) violations when the load suddenly increases. Our vision of continuous decentralized autoscaling avoids these issues by giving individual service instances the power to make scaling decisions in a distributed fashion. Each instance performs self-monitoring and evaluates its state. The service instance initiates upscaling if it detects an overload or downscaling if its load is below a specified threshold. By randomly determining scaling timing, we achieve quasi-continuous scaling behavior when multiple service instances are deployed. We discuss challenges regarding analytical modeling, simulation, and real-world evaluation of this approach.**

***Index Terms*—autoscaling, self-management, probabilistic approach, microservices, cloud computing**

I. Introduction

Autoscaling is one of the key challenges in modern container clouds. Numerous researchers have developed autoscaling approaches over the past decades, ranging from threshold-based methods to deep learning-based approaches [1]. However, our previous work [2] showed that many research autoscalers are not adopted in practice and that flexible autoscaling approaches are demanded.

Traditional autoscalers usually have two things in common: i) One autoscaler is usually responsible for a group of service instances. For microservices, horizontal scaling is mainly used, i.e., the number of service replicas is increased or decreased during scaling. Scaling decisions are made based on metrics that are averaged over all instances. ii) Scaling decisions are made at fixed periods. Often, autoscalers offer the possibility to set a scaling interval or cooldown times. Accordingly, the state of a service is evaluated at regular intervals. In summary, with traditional autoscaling, scaling decisions for a group of service instances are made by a central instance at fixed times. This has the following drawbacks: First, there is a single point of failure. No scaling occurs if the autoscaler, or the associated monitoring, fails or is incorrectly configured. Second, reaction times are already limited by design since decisions occur in fixed intervals. Choosing this interval too large can lead to violations of service level objectives (SLOs) in case of rising loads; too small intervals can lead to rapid up- or downscaling behavior [3].

Instead of viewing autoscaling as a problem in which the optimal number of instances must be decided at discrete points in time, we envision autoscaling as a continuous process. In our approach, individual service instances make independent scaling decisions, forming an overall scaling behavior. Comparable approaches for P2P architectures [4] and stream processing applications [5], [6] already showed the concepts' potential. Because the instances make the decisions at different times, there is a high frequency of scaling decisions. Our approach converges into a quasi-continuous process if many service instances are active. Random events play a critical role in this process, as not only the time of a scaling decision is determined randomly, but also the scaling decision is made based on probabilities in addition to instance-local monitoring metrics.

II. Proposed Approach

Our approach puts the task of autoscaling in the hands of the applications. We consider autoscaling as a MAPE loop, similar to previous works [4], [7]. In traditional autoscaling, the analysis, planning, and execution tasks lie with the autoscaler, as Fig. 1 shows. The service instances only expose monitoring values. In decentralized autoscaling, the instances additionally perform the analysis and planning steps and only pass their scaling decisions to an executor component. The existence of this component has only technical reasons. Communication with a control instance is necessary to perform a scaling decision in a cloud environment. For example, a scaling decision in Kubernetes must be handled via the control plane. If the instance itself had to execute a scaling action, it would need access to the control plane, requiring additional libraries and authentication. The decentralized autoscaling approach assumes that service instances can collect and process their own monitoring data. This is in line with modern monitoring libraries like Micrometer [8], which are often used to expose monitoring data at specific endpoints.

In the decentralized approach, each service instance executes the scaling algorithm depicted in Algorithm 1. After collecting recent monitoring data, it is checked whether metrics violate any SLOs. Depending on the result, it is evaluated whether upscaling or downscaling should take place. After the decision, a waiting time passes until the process is started again. We draw the waiting time as a random sample from a predefined distribution W. Constant waiting times would,

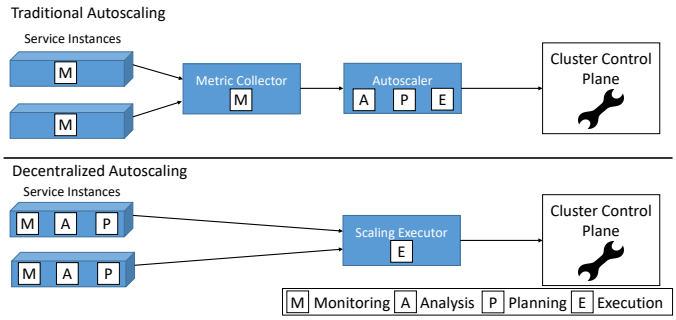

Fig. 1. Traditional and decentralized autoscaling.

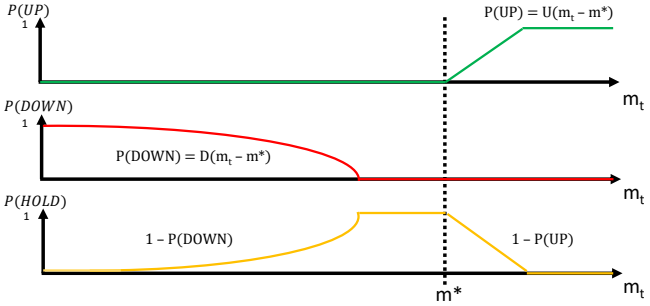

Fig. 2. Visualization of the scaling behavior for one instance.

in theory, lead to all instances making their decision almost simultaneously, which could lead to rapid up- or downscaling behavior [3]. Random waiting times lead to a near-continuous scaling process [9], [10], especially for many instances, as nearly every time, one instance makes a scaling decision.

Algorithm 1 Algorithm for Decentralized Autoscaling

Input: SLOs m^*, $D, U : \mathbb{R} \to [0; 1]$, Distribution W

while instance is running **do**
  Collect recent monitoring data into m_t
  Draw sample r from a uniform distribution in $[0; 1]$
  if m_t violates SLOs **then**
    $P(UP) = U(m_t - m^*)$
    if $r < P(UP)$ **then** decision = UP
    else decision = HOLD
  else
    $P(DOWN) = D(m_t - m^*)$
    if $r < P(DOWN)$ **then** decision = DOWN
    else decision = HOLD
  end if
  Submit decision to executor
  Draw sample w from W and wait for time w
end while

In every iteration, an instance chooses one of the three action alternatives: upscaling (UP), downscaling (DOWN), or no action (HOLD). We propose the following concept for decision-making: Let M be a scaling metric with a desired upper threshold m^* and let m_t be the current value of M at a scaling time t. We now consider the deviation $\Delta m = m_t - m^*$. If Δm is positive, the current measured value is above the set threshold, so we may want to scale up. The probability $P(UP)$ for an upscaling action is calculated based on a function U, which takes the value of Δm as input. U should be a monotonically increasing function, meaning that the larger Δm, the higher the upscaling probability $P(UP)$. Similarly, if Δm is negative, we consider a function D that calculates the downscaling probability $P(DOWN)$. We then make the final decision by drawing an equally distributed random number from the interval $[0; 1]$ and comparing it to either $P(UP)$ or $P(DOWN)$. Fig. 2 visualizes this principle. If m^* is a lower threshold, the upscaling or downscaling routines must be swapped, depending on the sign of Δm.

III. Research Aspects

In the following, we describe the theoretical and practical research aspects of this project.

Modeling: An analytical assessment of this approach is complex. The main research questions are: Under what conditions, e.g., for which functions U and D, is the system stable, i.e., for a constant load also a constant number of instances is supplied? What are the requirements for the scaling metric M? Secondly, we look at the scaling dynamics. For example, what is the (expected) time until the optimal number of instances is reached in case of a sudden load peak? We plan to apply methods of discrete-time analysis. Challenges include that instances are dynamically replicated and removed in the process and that the (expected) time between two scaling decisions within the system depends on the number of deployed instances. We plan to build on models from network survivability research [11].

Simulation: The analytical model is complex and is unlikely to be able to handle dynamic workloads. Therefore, it is essential to pursue a simulation-based approach for model validation. In addition, simulation allows testing any configuration of the model parameters U, D, and more. We might also include other parameters, such as the startup time of service instances, in our analysis. We plan to develop a discrete event simulation for this purpose.

Real-World Evaluation: Ultimately, the insights gained from modeling and simulation must also be applied in practice. Critical questions for the real-world evaluation of a prototype are: What metrics are available and reliable for scaling? Are there problems arising with real microservice applications? How does the approach compare to traditional autoscalers?

IV. Conclusion

In this paper, we presented our vision for a new autoscaling approach. It is based on three key features: (i) Decentralism: service instances make individual scaling decisions. (ii) Probabilism: Decisions and decision times are determined probabilistically. (iii) Continuity: Distributed decisions generate high decision frequencies, converging towards a near-continuous scaling process for many instances. This project

promises exciting results both from a theoretical and a practical point of view and aims to eliminate some conceptual drawbacks of traditional autoscalers.

REFERENCES

[1] P. Singh, P. Gupta, K. Jyoti, and A. Nayyar, "Research on auto-scaling of web applications in cloud: survey, trends and future directions," *Scalable Computing: Practice and Experience*, vol. 20, no. 2, 2019.

[2] M. Straesser, J. Grohmann, J. von Kistowski, S. Eismann, A. Bauer, and S. Kounev, "Why is it not solved yet? challenges for production-ready autoscaling," in *Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering*, ser. ICPE '22. New York, NY, USA: Association for Computing Machinery, 2022.

[3] M. Straesser, S. Eismann, J. von Kistowski, A. Bauer, and S. Kounev, "Autoscaler evaluation and configuration: A practitioner's guideline," in *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '23. New York, NY, USA: Association for Computing Machinery, 2023.

[4] N. M. Calcavecchia, B. A. Caprarescu, E. Di Nitto, D. J. Dubois, and D. Petcu, "Depas: a decentralized probabilistic algorithm for auto-scaling," *Computing*, vol. 94, 2012.

[5] M. M. Belkhiria and C. Tedeschi, "A fully decentralized autoscaling algorithm for stream processing applications," in *Euro-Par 2019: Parallel Processing Workshops: Euro-Par 2019 International Workshops, Göttingen, Germany, August 26–30, 2019, Revised Selected Papers 25*. Springer, 2020, pp. 42–53.

[6] G. R. Russo, "Towards decentralized auto-scaling policies for data stream processing applications." in *ZEUS*, 2018.

[7] M. S. Aslanpour, M. Ghobaei-Arani, and A. Nadjaran Toosi, "Auto-scaling web applications in clouds: A cost-aware approach," *Journal of Network and Computer Applications*, vol. 95, 2017.

[8] VMware Inc. (2023) Micrometer application monitoring. https://micrometer.io/.

[9] T. Hoßfeld, F. Metzger, and P. E. Heegaard, "Traffic modeling for aggregated periodic iot data," in *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, 2018.

[10] F. Wamser, P. Tran-Gia, S. Geißler, and T. Hoßfeld, *Modeling of Traffic Flows in Internet of Things Using Renewal Approximation*. Cham: Springer International Publishing, 2019.

[11] P. E. Heegaard and K. S. Trivedi, "Network survivability modeling," *Computer Networks*, vol. 53, no. 8, 2009, performance Modeling of Computer Networks: Special Issue in Memory of Dr. Gunter Bolch.