Challenges for getting research-based autoscalers production-deployed

Arami Alfarhani
Erik Wachtmeister

April 2023

We certify that generative AI, incl. ChatGPT, has not been used to write this essay. Using generative AI without permission is considered academic misconduct.

Contents

1 Introduction

The demand for many services can vary over time, software services included. As an example, a web server may occasionally need to handle a much higher load than usual. And at other times this server may have little to do at all. To be able to handle peak demand you might need to have a lot of running servers. But it would of course be wasteful and incur unnecessary costs if you were to run way more servers than needed most of the time. The issue is that of scaling the used resources, up or down, depending on the demand. Autoscalers have been developed to handle such issues automatically and are commonly used in cloud services [1].

1.1 Autoscaler internals

In broad terms, how do autoscalers work? [1] categorizes autoscalers by three fundamental approaches; reactive, proactive and hybrid. Autoscalers using the reactive approach make decisions on the observed load using measures retrieved from the running servers, for example, CPU and memory usage. Measures may also be drawn from the running application itself [1]. However, reactive based approaches might not be suitable for all types of applications, for example, not for services experiencing sudden spikes of demand [2]. The problem is that reactive methods need to base their decision on a average over time, and if the time window is too short, scaling might osccilate too much [3]. Therefore, it could be difficult to act in time. For cases were reactive methods will be difficult to use one might want to try one of the other two approaches. Proactive methods, [1], instead of waiting for the load to be high, try to predict when demand will be high or low and prematurely act on that prediction. The third approach, hybrid method, as implied, is some combination of the above.

1.2 Vertical vs. horizontal scaling

If one has come to the conclusion that scaling of the resources needs to be done, there are two main ways of how to perform the scaling [3]. One is to increase/decrease the allocated resources for the currently used entities. This approach is called vertical scaling. The other approach is horizontal scaling. It instead creates more entities, for example, servers. See figure 1.

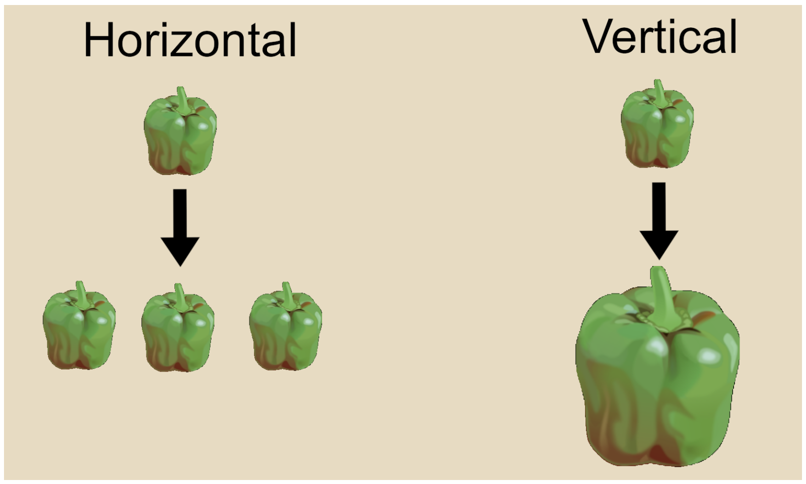

Figure 1: Horizontal vs vertical scaling; bellpeppers representing compute-units

1.3 Gap in current autoscalers vs. research based

Looking at some of the major cloud providers, it is possible to use autoscaling in many of their services [4][2][5]. For the Google BigTable service the available measures are CPU and memory utilization and a range with the minimum and maximum number of entities that can be launched [2]. Looking at services in [4] [5] more options are present. Some require setup of user-defined rules for when to scale, whilst others, like predictive-based scaling for AWS EC2, require less manual intervention and are easier to set up. Looking at [1], a survey of autoscalers for web application, it is evident that there exist many academic-research autoscalers. [6] have compared autoscalers used in practice with those from the research community. They noted that the autoscalers used in practice were often far simpler than those from the research community.

2 Challenges with autoscalers

The gap between practically used and research based autoscalers made Straesser et al. [6] question what makes it difficult to develop and adopt new autoscalers. Below, we summarize what we think were their most important thoughts on this and also comment and extend upon these using our own perspective and related research when applicable.

2.1 Balance of reactive and proactive autoscaling

One challenge set out by Straesser et al. is the problem of balancing reactive scaling with proactive scaling, in autoscalers which implement both paradigms, so-called hybrid scaling[6]. Reactive scaling is the process in which, as described, the system is afforded resources in accordance with the current load on the system. One way this can be implemented is by monitoring the current CPU and memory usage, and when a certain threshold has been reached allocate more of the used resource to that particular system[7]. A problem with only using reactive scaling is the delay between the signal of resource acquisition and freeing, and the actual allocation or deallocation of them, which, depending on the autoscaler being used can be costly and be ill-fitted for the current resource-requirements of a system. Another issue is possible over-allocation of resources, which is especially prevalent for threshold-based reactive scaling, which necessitates over-allocation to some degree. The other approach, being proactive scaling, attempts to allocate resources based on forecasted usage. These are often build on top of a statistical model, such as LSTM-models, which based on a history of usage metrics can allocate resources as a function of time[7]. By combining these two paradigms we are able to reliably manage resources over time without immediate feedback from the system by means of proactive scaling, while at the same time benefiting from the flexibility of reactive scaling, e.g. in cases of sudden surges in required resources.

The issue at hand concerns in which way these paradigms should be combined in order to best manage resources. This is as the decision outputs of the two methods often can come into conflict without any obvious way to resolve it. One way to do this is to delegate the scaling methods to upscaling and downscaling respectively, thereby preventing conflicts between the two and allowing the methods to be utilized more effectively[8]. Making use of this method of preventing conflicts Ali-Eldin et al. found that out of nine different configurations, delegating up and downscaling to reactive and proactive engines respectively was the most effective, improving SLA violation rates, the rate at which service level agreements for clients are broken, by two to ten times[8]. Ultimately, the way to combine the two methods effectively is far from a settled science and has a plethora of possible implementations, many of which have been proposed in papers showing their effectiveness in experimental settings[9].

2.2 Basing actions on the right metrics

For an autoscaler to function it needs to be given some measures that it can base its decision on whether to not interfere, scale up or scale down the used resources. However, reading from [1], there are many different such measures, and deciding on which ones to use is not straightforward [6]. The metrics can mainly be grouped into platform- and application-specific measures. Platform measures include what can be inferred from the platform that is running the application; for example CPU and memory usage. Application-specific measures on the other hand, as the name implies, are measures derived from the running application. However, application-specific does not necessarily mean that the metric is only available on that application. Many applications are written on top of common frameworks, and from these frameworks, one might collect semi-general measures, like in the Spring framework that was used in the study of [6]. One might also add additional monitoring packages to the application [4].

While platform-specific measures like CPU and memory usage are common measures for autoscalers, other measures may be better [6]. The problem with finding the right measures seems to come from the fact that different applications can have very different performance profiles. Comparing applications like, for example, ticket booking systems, which occasionally will experience very high spikes of demand appearing very rapidly in contrast to large computer simulations, it is easy to see that applications can behave very differently. And thus it follows naturally that some approaches will work better for some types of applications.

In [6] study, it is shown that combinations of platform- and application-specific measures seem most promising for creating good autoscalers, at least for reactive methods. Looking at proposed methods in the literature, the authors believe that autoscalers could become better if they included more application-specific measures. However, this consequently leads to it being more difficult to construct general autoscalers that could be used with any application.

2.3 Issues with configuration

Straesser et al. states that a problem that is necessary to address is the configuration required for autoscaling to operate effectively. This configuration can often take the form of for example setting thresholds for reactive scaling to operate around or parameters for engines utilizing statistical models. The main problem is that correctly configuring autoscaling for a service can be difficult in a way that it is properly utilized for the specific service being used[6]. This is compounded on the fact that the needs for a service can vary wildly, not only between different services, but even between different versions of the same service. This takes resources in initial costs in development, and the cost of maintenance required in reconfiguring the parameters in the event of changes in code that introduce changes in performance. This difficulty comes with uniquely problematic issues in the field of DevOps. A 2019 survey on industrial practices regarding performance within DevOps workflows concluded that around two thirds of participants did not conduct performance evaluations at all[10]. This has grave implications for the complex configuration of autoscalers, as this tool which very purpose is to optimize performance may be underutilized due to a lack of evaluation. The lack of evaluation also risks hampering reconfiguration of the autoscaler, as no concrete metrics to base the reconfiguration around are provided.

One way of addressing this issue, Straesser et al. argues, is for autoscalers to try to minimize the overhead caused by configuration and re-configuration, and instead focus on self-optimization. One approach of this is to use statistical models for self-configuration, such as using Kalman filtering, to minimize the manual configuration[11]. While solutions such as these still need some manual configuration, the overhead is smaller, thus lowering the threshold for developers to effectively utilize the software.

2.4 Quality of metrics

For autoscalers to operate properly, most models require accurate metrics from the applications and platform to be accurate and complete; this is the premise autoscaling works from. However, we face the risk of metrics being just the opposite, especially during high loads, which can compromise the operations of the scaling to be ineffective or downright counterproductive. This is especially problematic for application metrics, as they are "prone to delay or failed measurements", as the metrics need to be gathered by the application being encapsulated[6].

Another problem that often comes up is the issue of response times being used as a primary metric for many state-of-the-art autoscalers[6]. Straesser et al. argue that this is inherently problematic as basing scaling on response times is by definition prone to delay, as high response times only can be measured after said response time has passed. This can be worsened with systems built up by multiple services, where this high response time can propogate through the system, causing massive disparities between the current needs of the system and the needs as determined by the autoscaler. However, there is a 2023 paper suggesting that this latency can be used in finding bottlenecks within a greater system, and provide more accurate scaling[12]. The paper states that they seek to create a "bottleneck-aware autoscaling framework designed to prevent performance degradation", which would mitigate the degradation caused by for example propogating reaction times, if utilized correctly. The technology proposed is however very much conceptional, and not production-ready.

3 Reflection

All in all, all these issues are by no means exhaustive and there are plenty of issues that need to be addressed. Initially when learning that research-based approaches are not being implemented in practice one might think that a reason might be that others do not bother to implement them due to their complexity. Configuration overhead, as described, is a problem and it would be even worse if there were more to configure. This we deem to be the most important issue brought up by[6]. However, that might not be the most important issue over all. According to [13] the performance of research autoscalers seem mostly evaluated in custom experiments. It would be much better if evaluation was done on widely accepted benchmarks, although such benchmarks might be difficult to create. But without easy comparison, it might be difficult to convince others of a new and supposedly better autoscaler.

4 Conclusion

To conclude, instead of the more complex autoscalers being developed by researchers, much more simpler ones seem to be in practical use. That has lead to questions around what challenges one must overcome for getting research-based autoscalers adopted in production use. We presented challenges that newer methods hopefully will solve, at least partially.

We conclude that among the issues autoscaling faces, making them foolproof is paramount!

References

[1] Parminder Singh, Pooja Gupta, Kiran Jyoti, and Anand Nayyar. Research on auto-scaling of web applications in cloud: Survey, trends and future directions. *Scalable Computing: Practice and Experience*, 20(2):399–432, 2019.

[2] Cloud bigtable documentation: Autoscaling.

[3] Martinekuan. Autoscaling guidance - best practices for cloud applications.

[4] Mimckitt. Overview of autoscale with azure virtual machine scale sets - azure virtual machine scale sets.

[5] AWS User Group India. *Auto Scaling on AWS*. YouTube, Jun 2022.

[6] Martin Straesser, Johannes Grohmann, Jóakim von Kistowski, Simon Eismann, André Bauer, and Samuel 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*, ICPE '22, page 105–115, New York, NY, USA, 2022. Association for Computing Machinery.

[7] Li Ju, Prashant Singh, and Salman Toor. Proactive autoscaling for edge computing systems with kubernetes. *CoRR*, abs/2112.10127, 2021.

[8] Ahmed Ali-Eldin, Johan Tordsson, and Erik Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *2012 IEEE Network Operations and Management Symposium*, pages 204–212, 2012.

[9] André Bauer, Nikolas Herbst, Simon Spinner, Ahmed Ali-Eldin, and Samuel Kounev. Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field. *IEEE Transactions on Parallel and Distributed Systems*, 30(4):800–813, 2019.

[10] Cor-Paul Bezemer, Simon Eismann, Vincenzo Ferme, Johannes Grohmann, Robert Heinrich, Pooyan Jamshidi, Weiyi Shang, André van Hoorn, Monica

Villavicencio, Jürgen Walter, and Felix Willnecker. How is performance addressed in devops? In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, ICPE '19, page 45–50, New York, NY, USA, 2019. Association for Computing Machinery.

[11] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC '09, page 117–126, New York, NY, USA, 2009. Association for Computing Machinery.

[12] Shuaiyu Xie, Jian Wang, Bing Li, Zekun Zhang, Duantengchuan Li, and Patrick C. K. H. Pbscaler: A bottleneck-aware autoscaling framework for microservice-based applications, 2023.

[13] Martin Straesser, Simon Eismann, Jóakim von Kistowski, André Bauer, and Samuel Kounev. Autoscaler evaluation and configuration: A practitioner's guideline. *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, 2023.