Self-managed services using MirageOS unikernels

Al Amjad Tawfiq Isstaif
aati2@cam.ac.uk
University of Cambridge

Abstract

Next generation service-based systems will become increasingly finer-grain, larger-scale with heterogeneous performance characteristics. These services will need to be orchestrated across multiple IT infrastructures with application-specific auto-scaling capabilities to self-adapt to workload and environment changes. Today's generic autoscalers fail to address the adaptation needs of such systems and current methods for developing effectively application-aware autoscalers require substantial efforts and are far from practical. In this PhD project, I propose that distributed tracing alongside library operating systems (unikernels) can provide a cost-effective alternative approach to build self-scaling services. Such services will embody their own auto-scaling systems that are both generic and application-aware.

CCS Concepts: • **Computer systems organization → Cloud computing**.

Keywords: self-scaling, auto-scaling, elasticity, MirageOS, unikernels, microservices, distributed tracing, lightweight virtualization

ACM Reference Format:
Al Amjad Tawfiq Isstaif. 2020. Self-managed services using MirageOS unikernels. In *21st International Middleware Conference Doctoral Symposium (Middleware '20 Doctoral Symposium), December 7–11, 2020, Delft, Netherlands.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3429351.3431748

1 Background and motivations

MirageOS [15] is a library operating system that allows network applications to be written using high-level source code and compiled as single-purpose lightweight virtual machines that run on commodity hypervisors powering today's cloud platforms. With minimal memory footprints and fast startup times, unikernels effectively represent an efficient and secure alternative to release individual services compared to traditional virtual machines. However, the end-to-end management of a unikernel service-based system remains an independent problem.

Current practices for developing service-based systems are resulting in autonomous development teams producing large number of microservices with heterogeneous performance characteristics [10]. The increasing adoption of function as a service (FaaS) and serverless cloud computing is only leading to a finer-grain sizes of these services with even more fragmented state. Furthermore, traditional data center and cloud environments will no longer be the single place where these services will run. With the emerging IT infrastructures such as of fog and edge, tomorrow's services will need to be orchestrated across multiple data centers and consolidated on resource-constrained machines [14].

These various trends associated with the microservices approach raise the need for application-specific auto-scaling and orchestration capabilities so that these services would be able to self-adapt to workload and environment changes. Building effective generic autoscalers requires the consideration of the diversities of cloud applications in terms of their workload characteristics and resource requirements. Today's generic autoscalers fail to effectively address the adaptation needs of such systems and current methods for developing specialized autoscalers require substantial efforts that are not currently adopted in industry [17]. Therefore, achieving cost-effective autoscaling and orchestrating for such service-based systems, in both of traditional and emerging IT infrastructures, is still considered an open research area [10, 18, 20].

2 Challenges of autoscaling microservice systems

Autoscaling systems aim to achieve efficient resource allocation for cloud applications through on-demand acquisition and release of resources in response to workload changes. Rule-based auto scaling is the simplest and most widespread approach to autoscaling [18], where scaling decisions are triggered based on certain rules that declare thresholds of scaling indicators specified by the application provider. In the context of large-scale service-based systems, manually setting scaling indicators requires extensive experimentation and profiling of the application under a representative workload, which is a very costly process even when considering a modest degree of freedom in terms of workload and deployment options [2]. Even when such extensive experimentation is possible, statically-set thresholds for scaling

Middleware '20 Doctoral Symposium, December 7–11, 2020, Delft, Netherlands

ACM ISBN 978-1-4503-8200-7/20/12.
https://doi.org/10.1145/3429351.3431748

indicators are vulnerable to changes in microservice characteristics due to workload changes or new software releases [4, 5].

Furthermore, since the process of provisioning cloud resources requires some considerable time (e.g. startup time of a VM), this raises the need for proactive policies that predict workload change within a sufficient time frame, so that scaling decisions could take place before a QoS violation would occur [3]. This also includes estimating the amount of resources to be acquired or released when such autoscaling decisions are to be triggered. Runtime performance and workload models form the basis to develop such proactive policies. However, developing effective model-based self-adaptive strategies is a non-trivial task and hardly practical [17]. Recent survey [18] of the literature of auto-scaling of cloud applications, is dominant by the suboptimal approach of considering the scaling requirements of a single service or application tier, with very little number of works that consider the problem of effectively autoscaling of the service-based system as a whole.

3 Research proposal

Building on the successes of distributed tracing in addressing resource management tasks for cloud applications [12, 19, 21], I propose that unikernels extended with appropriate distributed tracing capabilities represent a promising building block to address the challenges of developing and deploying finer-grain and larger-scale service-based systems that self-adapt to changes in workload and operation environment. This unique combination will allow to build auto-scaling management systems that are both generic and application-aware as well as embodied within the service system. Such an approach has the potential to provide a practical and cost-effective alternative to current approaches for building effective autoscaling systems [17].

To the best of my knowledge, this proposal is the first attempt to explore the combination of unikernels with appropriate distributed tracing to develop efficient self-scaling microservices. On one side, distributed tracing can help tackle the problem of dynamic and real-time detection of resource bottlenecks across the complex and distributed structure of microservice-based systems in order to trigger effective scaling decisions that resolve these bottlenecks. On the other side, with the smaller granularity and the faster startup time of unikernels, resources can be provisioned reactively just-in-time [13], based on local service-level decisions, and in fine grains. This may lead to generic reactive autoscaling policies that are as effective as proactive policies, avoiding the need for highly specialized performance and workload models of the underlying system [3].

A distributed tracing model for reactive scaling: Increased waiting time is the essential influence of resource bottlenecks on service requests [7]. As such, resource waiting times on various software and hardware resources along the request path will represent the fundamental measurements to be collected. The end-to-end distributed tracing approach to collect such measurements is essential to trigger efficient QoS-aware scaling decisions based on request-level metrics (e.g. latency). Throughout a whole system perspective, dominant waiting times would be identified and resolved through a number of reactive fine-grained resource increments, and resources would be allocated only when necessary (e.g. when a QoS guarantee is about to be violated). Additionally, the request-centered approach would be useful to avoid unncessary scaling decisions due to artificial bottlenecks in upstream services caused by overloaded downstream services [19, 21].

Unikernel-based architecture for self-scaling: In addition to their strong isolation properties, unikernels can be positioned as an extreme option in the spectrum of emerging techniques of lightweight virtualization and kernel minimization [16]. All these options can be effective in various degrees in the development of efficient reactive instantaneous autoscaling policies. Compared to the increasingly popular container-based service mesh microservice architecture [11] (e.g. Istio), it is possible with unikernels to embed the functionalities of side-car service proxies as well as self-management logic alongside application code within the boundaries of the same VM [9]. This philosophy of minimalism and dedpulication introduces an additional architectural advantage which has the potential to minimize the management overhead across the control and data paths of the service-based system. Such an overhead is becoming increasingly relevant with the multiple control and data planes in today's data center networking and next generation software-defined networks [1].

Conclusion: The overall aim of my research is to develop a qualitative understanding of key parameters and dynamics contributing to an effective instantaneous reactive self-scaling microservice architecture. This includes studying the implications of various factors including the speed of bottleneck detection, VM startup time, resource/service discovery and load balancing as well as the granularity of resource allocations and duration of cooling down times. A self-scaling architecture would be mainly evaluated against an appropriate elasticity benchmark [6] using a set of reference applications (such as [8]). I plan to start from Mirage unikernels as an extreme case of lightweight execution units and compare to a representative spectrum of deployment options including VMs, containers, and other approaches to kernel specialization. Based on such an evaluation framework, I will work to identify use cases where the "extreme" approach of unikernels might be useful, such as next generation edge or in-network services [20]. Such use cases would be distinguished with a highly variant and unpredictable

workload in addition to the need for high assurance in application functionality and self-management logic.

Acknowledgments

I would first like to thank my supervisor Professor Richard Mortier of the Systems Research Group at the University of Cambridge for his guidance and valuable support. I would also like to thank the anonymous reviewers whose comments and suggestions helped improve and clarify this manuscript. This work was funded in part by the BT/Huawei Compute First Networking project.

References

[1] Antichi, G., and Rétvári, G. Full-stack sdn: The next big challenge? *Proceedings of the Symposium on SDN Research* (2020).

[2] Avritzer, A., Ferme, V., Janes, A., Russo, B., Schulz, H., and Hoorn, A. V. A quantitative approach for the assessment of microservice architecture deployment alternatives by automated performance testing. In *ECSA* (2018).

[3] Bauer, A., Herbst, N., Spinner, S., Ali-Eldin, A., and Kounev, S. Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field. *IEEE Transactions on Parallel and Distributed Systems 30* (04 2019), 800 – 813.

[4] Gan, Y., and Delimitrou, C. The architectural implications of cloud microservices. *IEEE Computer Architecture Letters 17* (2018), 155–158.

[5] Gotin, M., Lösch, F., Heinrich, R., and Reussner, R. H. Investigating performance metrics for scaling microservices in cloudiot-environments. In *ICPE '18* (2018).

[6] Herbst, N., Krebs, R., Oikonomou, G., Kousiouris, G., Evangelinou, A., Iosup, A., and Kounev, S. Ready for rain? a view from spec research on the future of cloud metrics. *ArXiv abs/1604.03470* (2016).

[7] Jiang, D., Pierre, G., and Chi, C. Autonomous resource provisioning for multi-service web applications. In *WWW '10* (2010).

[8] Kistowski, J., Eismann, S., Schmitt, N., Bauer, A., Grohmann, J., and Kounev, S. Teastore: A micro-service reference application for benchmarking, modeling and resource management research. *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)* (2018), 223–236.

[9] Koleini, M., Oviedo, C., McAuley, D., Rotsos, C., Madhavapeddy, A., Gazagnaire, T., Skejgstad, M., and Mortier, R. Fractal: Automated application scaling. *ArXiv abs/1902.09636* (2019).

[10] Kratzke, N., and Quint, P.-C. Understanding cloud-native applications after 10 years of cloud computing - a systematic mapping study. *J. Syst. Softw. 126* (2017), 1–16.

[11] Li, W., Lemieux, Y., Gao, J., Zhao, Z., and Han, Y. Service mesh: Challenges, state of the art, and future research opportunities. *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)* (2019), 122–1225.

[12] Mace, J., Bodík, P., Fonseca, R., and Musuvathi, M. Retro: Targeted resource management in multi-tenant distributed systems. In *NSDI* (2015).

[13] Madhavapeddy, A., Leonard, T., Skjegstad, M., Gazagnaire, T., Sheets, D., Scott, D., Mortier, R., Chaudhry, A., Singh, B., Ludlam, J., Crowcroft, J., and Leslie, I. Jitsu: Just-in-time summoning of unikernels. In *NSDI* (2015).

[14] Madhavapeddy, A., Mortier, R., Crowcroft, J., and Hand, S. Multiscale not multicore: efficient heterogeneous cloud computing. In *Proceedings of the 2010 ACM-BCS Visions of Computer Science Conference* (2010).

[15] Madhavapeddy, A., Mortier, R., Rotsos, C., Scott, D., Singh, B., Gazagnaire, T., Smith, S., Hand, S., and Crowcroft, J. Unikernels: library operating systems for the cloud. In *ASPLOS '13* (2013).

[16] Manco, F., Lupu, C., Schmidt, F., Mendes, J., Kuenzer, S., Sati, S., Yasukata, K., Raiciu, C., and Huici, F. My vm is lighter (and safer) than your container. *Proceedings of the 26th Symposium on Operating Systems Principles* (2017).

[17] Mendonça, N., Garlan, D., Schmerl, B., and Cámara, J. Generality vs. reusability in architecture-based self-adaptation: the case for self-adaptive microservices. In *ECSA '18* (2018).

[18] Qu, C., Calheiros, R., and Buyya, R. Auto-scaling web applications in clouds: A taxonomy and survey. *arXiv: Distributed, Parallel, and Cluster Computing* (2016).

[19] Suresh, L., Bodík, P., Menache, I., Canini, M., and Ciucu, F. Distributed resource management across process boundaries. *Proceedings of the 2017 Symposium on Cloud Computing* (2017).

[20] Vaquero, L., Cuadrado, F., El-khatib, Y., Bernabé, J., Srirama, S., and Zhani, M. F. Research challenges in nextgen service orchestration. *Future Gener. Comput. Syst. 90* (2019), 20–38.

[21] Zhou, H., Chen, M., Lin, Q., Wang, Y., She, X., Liu, S., Gu, R., Ooi, B. C., and Yang, J. Overload control for scaling wechat microservices. *Proceedings of the ACM Symposium on Cloud Computing* (2018).