Model-based Resource Management for Fine-grained Services

Gias, Alim Ul

Model-based Resource Management for Fine-grained Services

Alim Ul Gias, University of Westminster

Homepage: https://sites.google.com/view/alimulgias/

Thesis: Model-based Resource Management for Fine-grained Services

Advisor: Giuliano Casale, Imperial College London

Brief Biography: Alim Ul Gias is currently a Research Associate at the Centre for Parallel Computing (CPC), University of Westminster. He completed his PhD from Imperial College London in 2022. Before starting his PhD, Alim was a lecturer at Institute of Information Technology (IIT), University of Dhaka (DU). He completed his bachelor's and master's program from the same institute. His current research focuses on different Quality of Service (QoS) aspects of cloud-native applications e.g., microservices. In particular, he aims to address the performance and resource management challenges concenrining the microservices architecture.

Research Summary: The emergence of DevOps has changed the way modern distributed software systems are developed. Architectures decomposed in fine-grained services, such as microservices or function-as-a-service (FaaS), are now widespread across many organizations. From a resource management perspective, although the systems built with such architectures have many benefits, there are still research challenges that need further attention. In this study, we have focused on three such challenges, each concerning a specific system resource: compute, memory, or storage.

Firstly, we focus on scaling the capacity of microservices at runtime. Here, the challenge is to design an autoscaler that can decide between vertical and horizontal scaling options to distribute the CPU capacity. Secondly, we focus on estimating the required capacity of an on-premises FaaS platform such that the service level agreements (SLAs) for function response times are satisfied. The challenge here is to address the cold start dilemma, i.e., that a cold start delays a function response but reduces the memory consumption. Thus, we must find a limit of cold starts such that the memory-consumption remains in-check while satisfying the SLAs. Finally, we focus on the storage management for distributed tracing targeted at microservices. The volume of such traces generated in a data center can be in the scale of tens of terabytes per day, but only a small fraction of these traces is useful for troubleshooting. The objective then is to sample only the useful traces.

The key to addressing all these challenges is first, modeling the dynamics concerning the resources and subsequently, leveraging the model in a resource controller. To address the first challenge, we have developed an autoscaler ATOM that leverages layered queueing network (LQN) models to take its scaling decisions. For the second challenge, we have developed COCOA, a cold start aware capacity planner. COCOA utilizes M/M/k setup and LQN models to assess the cold start scenario and estimate the required capacity. Finally, addressing the third challenge, we propose SampleHST, a trace sampler that works under a storage budget constraint. SampleHST relies on either bag of words or graph-based models to represent a trace and groups similar traces using online clustering to perform sampling.

We discuss the aforementioned contributions briefly in the following sections. We also discuss how these contributions amalgamate from a resource management perspective.

Auto-Scaling the Compute Capacity

In a microservice architecture, each of the services is lightweight in nature allowing a fine-grained resource allocation. For an instance, instead of allocating a single CPU core per service, half a CPU core can be allocated. Controlling capacity at this granularity reduces resource wastage compared to traditional virtualization mechanisms. However, such compute capacity allocation poses a dilemma when there is an increase in the current workload of the system. In such a case, the CPU capacity needs to be scaled automatically. Typically there are two possible choices - increase the CPU capacity of the container, which is referred as vertical scaling, or add another replica of the service, which is referred as horizontal scaling. This problem is more relevant when the services are inter-dependent, e.g., the microservices architecture, as the service communication pattern also have an effect over the scaling decision. This is similar to the well studied problem in queueing theory - one fast CPU versus several slow CPUs [4]. This problem is applicable for microservices as small fractions of CPU capacity can be allocated and the capacity can be changed both horizontally and vertically.

The research objective, in this case, is to develop a method that will scale the CPU capacity of microservices at runtime. To be more specific, the scaler should be able to assess different scaling options and suggest the optimal one. This can be further divided into two sub-goals.

- *Modeling Microservices:* The first goal will be to model a microservices application using LQN [2], which is already shown to be effective to model modern distributed systems [5]. The model should incorporate

the system dynamics of the application e.g., workload, shared resources, multiple application layers and provide different performance metrics, such as throughput and utilization, as the output.

- *Optimal Scaling:* The second goal is to provide optimal scaling decisions for microservices. In particular, there should be a controller that, using the LQN model, can assess different combinations of vertical and horizontal scaling configuration. Based on this assessment, it should select an optimal configuration for the current workload and eventually apply that on the system.

To address our first research objective, we have developed ATOM, a model-driven autoscaler tailored for microservices. We first demonstrate that if a service can be both vertically and horizontally scaled, the workload characteristics need to be taken into account to decide which one of the two scaling actions is preferable. ATOM incorporates this by leveraging layered queueing network models. It instantiates and solves, at run-time, such models of the application for different scaling configurations. Computational optimization is used to dynamically control the number of replicas for each microservice and its associated container CPU share, overall achieving a fine-grained control of the application capacity at run-time. Experimental results indicate that for heavy workloads ATOM offers around 30-37% higher throughput than baseline model-agnostic controllers based on simple static rules. We also find that model-driven reasoning reduces the number of actions needed to scale the system as it reduces the number of bottleneck shifts that we observe with model-agnostic controllers.

Capacity Planning considering Cold Starts

The functions in the FaaS platforms are usually implemented as lightweight services. As a result, the services have a quick start-up time. This opens-up the possibility of only loading a function when there is a continuous demand rather than always keeping it loaded in the memory. Removing a function from the memory reduces the memory consumption but this adds a latency delay for the upcoming service request. This is known as the cold start problem [7]. Even though the functions are light-weight and have a low start-up overhead, there can be cases when these cold start delays affect the function response time significantly. This poses a dilemma regarding installation of an on-premises FaaS platform. If a function is kept in the memory indefinitely, thus reducing cold starts, it increases the required memory capacity. On the other hand, allowing cold starts will affect the service-level-agreements (SLAs) for response time. To solve this problem, the dynamics among the SLAs, cold starts and memory consumption need to be assessed. Striking a balance among these factors will ultimately reduce the required memory capacity while satisfying the SLAs.

In this context, our research objective is to develop a capacity planning method that considers the trade-off between latency delay and conservation of primary memory due to cold starts. The method should identify a limit of cold starts such that the SLAs are satisfied, while reasonably limiting the memory consumption. This could be divided into the following steps.

- *Modeling Cold Starts:* The first challenge is to model the cold start scenario in a Function-as-a-Service (FaaS) platform. To be particular, the model should be able to estimate the response time considering a varying degree of function cold start delays and service times.
- *Capacity Planning:* Once we have a model of the cold start scenario, it needs to incorporated in a capacity planner. For a FaaS-based system to be deployed, with given function cold start delays and service times, the planner should provide a deployment configuration that includes the required CPU and memory capacity that satisfy the SLAs for response time.

To address our second research objective, we propose a novel approach, COCOA, to size an on-premises FaaS platform. We have initially investigated the similarity of this problem with the hit rate improvement problem in TTL caches and concluded that solutions for TTL cache, although potentially applicable, lead to over-provisioning in FaaS. COCOA addresses this over-provisioning problem. It uses a queueing-based approach, leveraging M/M/k setup class of models [3] and LQN, to assess the effect of cold starts on the function response times and sets the function idle periods to values such that the cold starts remain tolerable from the perspective of service level agreements (SLAs). Furthermore, it considers different memory consumption values depending on whether the function is idle or in execution. Based on this assessment, COCOA estimates the required CPU and memory capacity to serve the expected workload. Using an event-driven FaaS simulator, *FaasSim*, we have developed, we show that COCOA can reduce over-provisioning by over 70% in some workloads, while satisfying the SLAs for response time.

Trace Sampling with Storage Constraint

In a microservice ecosystem, the services are typically deployed and managed independently. These services are not necessarily deployed in a single server. Furthermore, as all these services can be scaled individually, they can have multiple replicas spanning across multiple servers, making a highly decentralized architecture. This independent deploying and scaling of the microservices, spanning across multiple servers, creates a problem from the perspective of maintenance. In particular, keeping the logs of a request workflow becomes difficult. Distributed tracing is an approach developed primarily to aid in this scenario. It profiles the user requests spanning across multiple microservices. For a data center hosting microservices in the scale of tens of thousands, the volume of traces generated could easily be in the scale of terabytes per day. To store this traces for future troubleshooting requires a large storage capacity. However, only a small fraction of these traces are helpful for such purposes. This opens up the possibility of reducing storage requirement by saving only the interesting traces in terms of, for example, identifying QoS violations. The challenge is to identify those interesting traces and sample according to a storage budget.

The research objective, in this context, is focused on saving distributed traces with a storage budget. Here, we aim to develop a trace sampler, with a limitation on the percentage of traces it can sample, that primarily focuses to save the interesting or anomalous traces. This involves two major challenges.

- *Anomaly Detection from Trace Stream:* The first challenge is to differentiate the interesting or anomalous traces from the normal ones. This will initially require

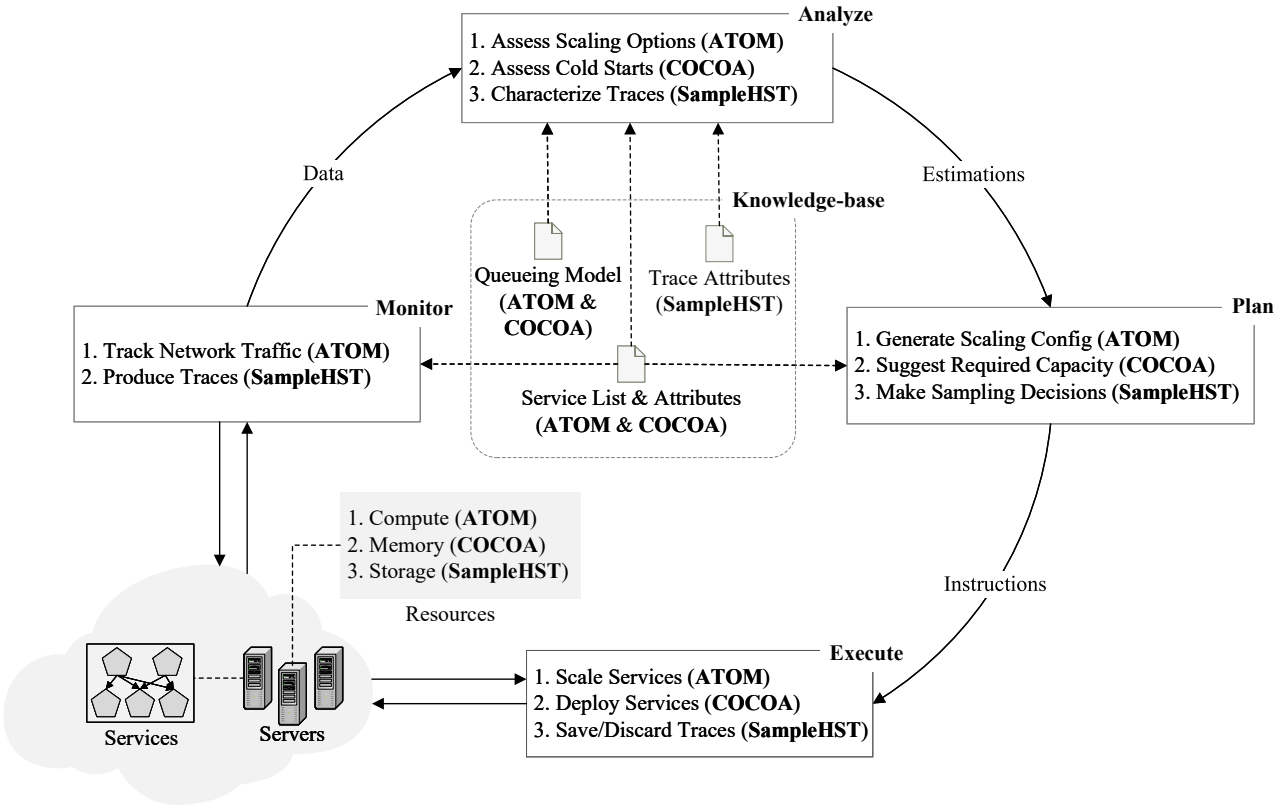

Figure 1: A bird's eye view of the research contributions from a resource management perspective. The contributions *i.e.*, ATOM, COCOA and SampleHST are presented as a sequence of different management tasks such as Monitor, Analyze, Plan and Execute. The focus of each contribution is on one of the following resources: Compute, Memory and Storage. The tasks leverage different artefacts such as queueing models, service attributes, trace attributes, etc. to achieve their objective.

a trace model and after that a classifier that will detect the anomalies in an unsupervised manner.

- *Trace Sampler:* The second challenge is to transform this anomaly detection process to a trace sampler. This requires to characterize the traces such the the anomalous and normal traces form distinct groups and depending on the budget, the traces will be sampled.

To address our final research objective, we propose SampleHST, a novel trace sampler for distributed tracing with a storage budget constraint. SampleHST is tailored to work on a stream of trace data and taking the sampling decision in an unsupervised manner. Depending on the definition of interesting traces, SampleHST uses either a Bag of Words (BoW) or a Graph model to represent a trace. This representation is then passed to a forest of Half Space Trees (HSTs) [6] for characterizing the trace with a mass score, which indicates the observed frequency of traces of similar types. These mass scores are used to cluster the traces using an online method based on a variant of the mean shift algorithm [1]. Finally, the association of each trace with a cluster, where certain clusters are prioritized over the others, is used to take the sampling decision. We have compared the performance of SampleHST with recently suggested methods using data reported in literature and produced in a data center. Depending on the properties of the experiment, we set the evaluation criteria using either precision, recall or F1-Score. The experiments demonstrate that SampleHST produces 1.2× to 19× better results than the recent methods.

Resource Managment Perspective

Although these contributions address a wide range of research challenges, in principle, they can be considered as a sequence of resource management tasks, such as monitor, analyze, plan and execute, which work on a shared knowledge-base. Most of these tasks are required for all the contributions *i.e.*, ATOM, COCOA and SampleHST. For example, all of them need to analyze a wide range of data by leveraging artefacts like a queueing model or the trace structure. In the case of ATOM and COCOA, the analysis involves assessing the scaling scenarios and the effect of cold starts. In the case of SampleHST, the analysis phase focuses on trace characterization such that the traces indicate anomalous system behaviors. We present the thesis contributions from this resource management perspective in Fig. 1. Note that, all the tasks are not associated with all the contributions. Here, COCOA is not associated with the monitor task as it focuses on capacity planning, rather than runtime management, which does not require an active monitoring component.

Representative Papers:

[1] ATOM: Model-Driven Autoscaling for Microservices (Accepted in ICDCS 2019) with G. Casale and M. Woodside

[2] COCOA: Cold Start Aware Capacity Planning for Function-as-a-Service Platforms (Accepted in MASCOTS 2020) with G. Casale

[3] SampleHST: Efficient On-the-Fly Selection of Distributed Traces (Submitted in NOMS 2023) with Y. Gao, M. Sheldon, J. A. Perusquía, O. O'Brien and G. Casale

1. REFERENCES

[1] Baruah, R. D., and Angelov, P. Evolving local means method for clustering of streaming data. In *Proceedings of the International Conference on Fuzzy Systems* (2012), IEEE, pp. 1–8.

[2] Franks, G., Al-Omari, T., Woodside, M., Das, O., and Derisavi, S. Enhanced Modeling and Solution of Layered Queueing Networks. *IEEE Transactions on Software Engineering 35*, 2 (2008), 148–161.

[3] Gandhi, A., Doroudi, S., Harchol-Balter, M., and Scheller-Wolf, A. Exact Analysis of the M/M/k/setup Class of Markov Chains via Recursive Renewal Reward. In *Proceedings of the SIGMETRICS* (2013), ACM, pp. 153–166.

[4] Harchol-Balter, M. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge Univ. Press, 2013.

[5] Shoaib, Y., and Das, O. Web application performance modeling using layered queueing networks. *Electronic Notes in Theoretical Computer Science 275* (2011), 123–142.

[6] Ting, K. M., Zhou, G.-T., Liu, F. T., and Tan, S. C. Mass estimation. *Machine Learning 90*, 1 (2013), 127–160.

[7] Wang, L., Li, M., Zhang, Y., Ristenpart, T., and Swift, M. Peeking Behind the Curtains of Serverless Platforms. In *Proceedings of the USENIX Annual Technical Conference (ATC)* (2018), USENIX, pp. 133–146.