Prediction-Guided Design for Software Systems

Si Qin[1], Yong Xu[1], Shandan Zhou[2], Qingwei Lin[1],
Hongyu Zhang[3], Dongmei Zhang[1], Saurabh Agarwal[2], Karthikeyan Subramanian[2],
Eli Cortez[2], John Miller[2], Chris Cowdery[2], Shanti Kemburu[2], Thomas Moscibroda[2]
[1]Microsoft Research, China [2]Microsoft Azure, USA [3]The University of Newcastle, Australia
[1,2]{siqin, yox, shazho, qlin, dongmeiz, saaga, karthik, eli.cortez, johnmil, chcowder, shkembur, moscitho}@microsoft.com
[3]hongyu.zhang@newcastle.edu.au

Abstract

While software system development is commonly conducted with explicit rules, machine learning (ML) has been driving a revolution in modern system design. In this paper, we introduce a new prediction-guided paradigm, which leverages ML techniques to support decision-makings for the system itself. In the proposed design, the system would be automatically driven by various type of data, e.g., system workloads, user behaviors, and platform operations, etc. More importantly, it brings a mindset of "proactive" to developers. Some significant issues can be thus eliminated before becoming catastrophe. In order to illustrate the benefits of the proposed paradigm, we present a project showcase, intelligent buffer management, which is used to achieve an optimal trade-off between having sufficiently large buffers to avoid failures and minimizing excess capacity in Microsoft Azure. It is designed in the prediction-guided paradigm to dynamically and proactively adjust the reserved buffer based on customer workload patterns and platform operations. The project not only significantly improves CFR (capacity fulfillment reliability) of tenant growth, but also reduces millions of dollars in COGS (cost of goods sold) for Microsoft.

Prediction-Guided Design

Currently, most of software system developments have been conducted based on human experience and comprehension. Given a task, domain knowledge is usually transferred into source code with explicit rules. After system evaluation, developers could gain more understanding about the system and could further improve the code. This process could go through several rounds, as illustrated in Figure 1.

However, this code-centric approach is often less optimal. On the one hand, in large-scale software systems, the internal details and the inter-dependencies of software modules could be extremely complicated. On the other hand, the software systems should be adaptive to different inputs and environments, and should satisfy the ever-changing business rules and requirements. Therefore, it is hard to frequently change the source code to cater for changes and the traditional code-centric approach is difficult to scale.

Recent advances in machine learning (ML) framework have ignited changes in system design. In view of the capabilities in learning complex correlations, causalities, and

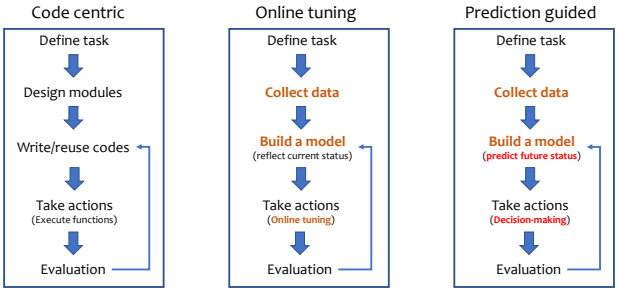

Figure 1: The different designs for software systems.

patterns among components from the data, ML-based approaches therefore hold the potential in building a bridge from the task definition to system operation directly, particularly in a dynamic environment without so many human inputs. In the data-driven context, software development would consist largely of developing a data processing pipeline and then streaming the corresponding output into an ML model to perform a given task (Ré 2018). As such, developers pay more attention to collect data and build a model instead of designing modules and writing codes. Likewise, large but complicated training datasets and cutting-edged models play more crucial roles in improvements of performance than the abundant domain knowledge and code efficiency. In practice, however, current prevailing designs limit the merits of ML to the "dynamic". Specifically, a couple of parameters are set and then can be online tuned by leveraging ML techniques based on workloads (Li and et al. 2018). While output values from the model are the best choice for the current status, they might not be optimal for the future. Such a reactive approach reacts the past rather than anticipate the future. We are still expected to have some sleepless nights when considerable issues come.

Besides intricate dependencies among components, ML can learn factors that impact system behaviors from historical events and incidents, and then make a fair prediction for the future system status. That allows the system to make a proactive response to eliminate significant issues before they become serious problems. In Fig. 1, we introduce our prediction-guided design as compared to the above-mentioned two paradigms. In the proposed paradigm, the ML model is utilized to predict future system status in

supporting decision-makings (Kruchten and et al. 2009; Zhang and Jarzabek 2005) for the system itself. There are three keywords: **data**, **status**, and **action**. The telemetry data are collected from various of perspectives (e.g., system workloads, user behaviors, and platform events, etc) with multiple temporal granularity (e.g., 15 minutes, hour, day, week, month, etc) and spatial granularity (e.g., location, version, operation system type, etc). The ML model exploits the data as the inputs and predicts the future status. A decision-making process will be triggered in view of the predicted status and then the system will take the corresponding action automatically. Different actions also have different impacts on data distribution, the model is thus necessary to capture these distinctions on data drifts. Note that the prediction-guided design not only brings the "dynamic" to developers, but also has a mindset of "proactive". In this paradigm, the system would learn lessons from the past and present, and make an appropriate choice for the future.

Example: Intelligent Buffer Management

A typical cloud serivce system contains a large number of physical servers, or "nodes". These nodes are arranged into racks and a group of racks form a cluster distributed in multiple regions. In Azure, a new deployment request is initiated when a customer requires an amount of specific type virtual machines (VMs) in a particular region. Afterwards, the allocator of the platform selects a few clusters in this region as candidates, and then previews the requested allocation on all candidates. Based on preview results, VMs are eventually deployed on the optimal cluster. Correspondingly, the customer is referred to as the tenant of the chosen cluster with a unique tenant id.

On the other hand, the cloud systems allow tenants to allocate and release VMs according to their demands. However, such a pay-as-you-go style makes resource demands more volatile. To improve CFR of further tenant growth, Azure maintains a buffer within each cluster, i.e., preserves an amount of capacity by preventing new deployments to land in this cluster in preview. As such, buffer capacity management is of fundamental importance in Azure. However, traditional approaches implement the same strategy on any cluster of similar size and property, regardless of the specific workload deployed within the cluster. In so doing, the buffer is necessarily set for the worst case because sufficient capacity needs to be maintained for all circumstances. This static and one-fits-all management therefore ends up preserving too much buffer capacity in a large number of clusters, resulting in low utilization and high COGS.

To overcome this problem, we design a new buffer management approach based on an intelligent admission control system (InACS). Following with the prediction-guided paradigm, the InACS dynamically and proactively adjust the reserved buffer by governing which new deployments can be admitted into the cluster, and which ones are not. The architecture of InACS is presented in Fig. 2. At the heart is an ML-based prediction-engine that monitors tenants that already deployed and predicts further growth demand resulted from these deployments within the cluster. The admission controller takes the prediction result as an input, and makes

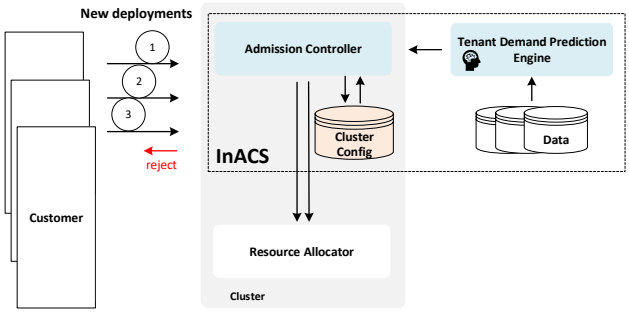

Figure 2: The architectural design of InACS.

a binary decision on whether or not allow VM new deployment to deploy into this cluster.

Specifically, we first collect related signals that show high correlations with tenant demand growth. The data can mainly be categorized into three aspects: tenant (e.g., VM type, core utilization, etc.), cluster (e.g., hardware type, cluster size, etc.), and platform (e.g., fragmentation, policy, etc.). The goal of the tenant demand prediction engine is to effectively predict whether a cluster will have "high", "medium", and "low" tenant demand growth in the next τ days, based collected feature in most recent T days. As such, it is a multi-class classification problem over a number of temporal and spatial features, whose labels are given by

$$\begin{cases} low, & \mu_i < \alpha\% \text{ and } \mu_c < \alpha\%, \\ medium, & otherwise, \\ high, & \mu_i > \beta\% \text{ and } \mu_c > \beta\%, \end{cases} \quad (1)$$

where μ_i and μ_c denote the maximum intra-day demand growth and maximum cross-day demand growth within next τ days respectively, and two threshold α and β are set based on the domain knowledge from production teams. In clusters with high predicted demand growth, the admission controller would apply a more restrictive policy and start reject new deployment earlier, thus preserve more capacity for future tenant growth. For those clusters with low predicted tenant growth, the admission controller would be more lenient in allowing new deployments into the cluster.

Our intelligent buffer management approach has been successfully integrated into the capacity pipeline of Microsoft Azure. So far, it achieves more than 95% precision and 60% recall. Compared with traditional approaches, InACS not only improves CFR of tenant growth, but also leads to millions of dollars reduction in COGS.

References

Kruchten, P., and et al. 2009. The decision view's role in software architecture practice. In *IEEE Software*.

Li, Z. L., and et al. 2018. Metis: Robustly optimizing tail latencies of cloud systems. In *Proc. of USENIX ATC*.

Ré, C. 2018. Software 2.0 and snorkel: Beyond hand-labeled data. In *Proc. of SIGKDD*.

Zhang, H., and Jarzabek, S. 2005. A bayesian network approach to rational architectural design. *International Journal of Software Engineering and Knowledge Engineering* 15:695–718.