RETROSPECTIVE: Bubble-Flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers

Jason Mars, Lingjia Tang
University of Michigan
{profmars, lingjia}@umich.edu

I. CONTEXT

The Bubble-Flux paper, published at the International Symposium on Computer Architecture (ISCA) in 2013, sought to address the challenge of microarchitectural resource contention between co-located applications and its effect on Quality of Service in data centers. During this period, the field of data center computing was in its nascent stages, with a particular focus on holistic Warehouse-Scale Computers (WSCs) architecture and design. Indeed, this was a time when technologies that have since become the norm in this domain, such as Kubernetes, were in their experimental phase and largely restricted to in-house operations at tech giants like Google.

During this era, the new performance indicator known as quality of service (QoS) had emerged in the realm of datacenter computing and was swiftly recognized as the key metric for evaluating performance in this domain. Ensuring the QoS for latency-sensitive applications was critical for optimizing server utilization and curtailing costs, especially when considering multiple user-facing applications run on the same server in WSCs. Bubble-Flux aimed to precisely predict contention and QoS at scale and in real time.

The conceptualization of Bubble-Flux, along with its predecessor Bubble-Up [1], was driven by Jason Mars over the course of two summers. Initially, as a research intern at Google, Mars formulated Bubble-Up, which was later published at Micro 2011 and was selected as a MICRO Top Picks in the community. Bubble-up represented a proof of concept the provided a controlled static profiling analysis for precise prediction. The technique was evaluated using production Google workloads. However Mars then developed the idea further into Bubble-Flux during a subsequent summer as a visiting scholar at Google after his appointment as an assistant professor at the University of California, San Diego (UCSD). Bubble-Flux was an real-time scale out solution that continuously ran on 1000s of machines as a service reporting contention and forecasting QoS as workloads change over the diurnal pattern in data-centers.

Historically, performance predictability in the context of microarchitectural resource contention had been an academic focal point since 2008 through a variety of techniques spanning compiler and OS solutions for isolation, among others. However, the distinguishing feature and key insight of Bubble-Flux and Bubble-Up was the recognition of the intractability of precise predictions using the modeling approach the community had been working on, given the inherent complexity of real world microarchitecture. Multiple applications can contend for numerous disparate microarchitectural resources, and these resources change every generation of processor released.

In response to this, Jason Mars, in collaboration with Lingjia Tang and colleagues, developed a range of novel techniques that are capable of operating at scale across thousands of machines, saving companies like Google millions of dollars in practice.

II. Key Challenges at the Time

The Bubble-Flux innovation was made on the heals of the early insights toward sample-based empirical approaches for precise real-hardware prediction. Instead of the modeling approach, Bubble-up [1] uses carefully crafted sensitivity profiler (bubbles) to stress test applications and generate a sensitivity curve. The curve is then used to precisely predict QoS degradation in colocation scenarios. However there were key limitations that created a large gap to bridge to realize deployable and salable solutions.

The first of these key limitations was the assumption inherent in solutions at the time such as Bubble-Up that necessitated a priori knowledge of applications. This presumption was a significant constraint on the methodology's general applicability and robustness. These works were limited in their ability to handle new scenarios where batch applications were scheduled but had not been profiled previously. In the (relatively common) scenario that a new application is deployed alongside long running latency sensitive applications, these approaches that require prior knowledge couldn't guarantee a precise Quality of Service (QoS) during co-location.

This limitation underscored the inherent inflexibility of such approaches, and the reliance on prior profiling left them ill-equipped to deal with novel or dynamic scenarios, laying the groundwork for the development of Bubble-Flux.

A second key limitation presented itself in inability of prior work to adapt to the dynamic phase-level behaviors of applications. This lack of adaptability drastically curtailed potential improvements in utilization realizable and raised the risk of QoS violations when the application's behavior diverged from its profiled behavior unexpectedly. Take, for example, the dynamic behavior of load fluctuations in latency-sensitive applications such as Web-search, social networking, or email services. These applications often experience significant load fluctuations, typically due to patterns in user behavior. For instance, an email service might face a high load at 8AM on Mondays but a significantly lower load at 3AM. Moreover, besides load fluctuations, changes in input latency-sensitive and batch applications may alter the QoS sensitivity of the latency-sensitive application or the pressure score of the batch application. These challenges catalyzed the need for a more adaptable and dynamic approach, subsequently leading to the creation of Bubble-Flux.

A third significant limitation of prior work at the time was in their scope only covering pairwise co-locations, that is, its inability to deal with scenarios where more than two applications are co-located. This limitation indicates a fundamental lack of scalability of static profiling techniques for QoS prediction; there is a state explosion as you scale applications.

The sum of these limitations - the requirement for a priori knowledge, the lack of adaptability to dynamic behavior changes, and the restriction to pairwise co-locations - significantly impeded the broad applicability of static approaches. Recognizing these shortcomings, we posited the necessity for a dynamic yet empirical approach that could address these issues. This approach needed to capture real-time QoS sensitivity and precisely manage QoS online, leading to the evolution of Bubble-Flux from the constrained Bubble-Up methodology.

III. The Bubble-Flux Solution

Our response to these challenges was the development of Bubble-Flux, a runtime system encapsulating a novel approach to application co-location. Bubble-Flux, which functioned as a user-level daemon running on each server, consisted of two main components: the Dynamic Bubble and the Online Flux Engine.

The first of these, the Dynamic Bubble, had a singular objective - to conduct an online, lightweight characterization of the latency-sensitive application. By measuring the instantaneous QoS sensitivity of the application, it could precisely predict potential QoS interference due to co-location. It facilitated the identification of "safe" co-locations to optimize utilization. The Dynamic Bubble operated by spawning a memory bubble probe as needed, incrementally applying pressure on the memory subsystem and observing how different levels of pressure affected the QoS of the latency-sensitive application. The Flux Engine controlled the performance interference generated by the bubble probe itself, ensuring accurate sensitivity curves while keeping the characterization overhead to a minimum.

When scheduling batch applications, the cluster scheduler employed a Dynamic Bubble to retrieve the instantaneous QoS sensitivity curve of the latency-sensitive application running on a given server. This sensitivity curve was essential for precise QoS interference prediction, enabling the scheduler to map batch applications effectively.

The second component of Bubble-Flux was the Online Flux Engine, which utilized a phase-in/phase-out (PiPo) mechanism to dynamically enforce application QoS requirements. PiPo relied on lightweight online QoS monitoring and precise QoS management to adaptively control the execution of batch applications. It could manage multiple applications, scaling beyond pairwise co-locations. This capacity was particularly helpful when dealing with first-time batch applications, for which predictions could not be made.

The Flux Engine provided a safety net against potential QoS mis-predictions or unexpected dynamic behaviors that rendered previous predictions irrelevant. In situations with no safe co-location available, the Flux Engine could further increase utilization by running partially phased-out batch applications. When encountering unknown applications, the Flux Engine could improve utilization without QoS violations by gradually phasing in the application.

The Dynamic Bubble and the Online Flux Engine worked synergistically. The QoS prediction provided by the Dynamic Bubble facilitated more intelligent co-locations of compatible applications so that aggressive phase-outs could be minimized or even avoided. On the other hand, the Flux Engine provided a safety mechanism to handle potential QoS mis-predictions or unexpected dynamic behaviors, ensuring a balance between maintaining QoS and improving server utilization.

IV. The Heritage in 2023

Reflecting upon our journey in the development of Bubble-Flux, we are struck by the magnitude of its influence and the depth of its implications. When Bubble-Flux was conceived, it was a pioneering approach in its space, the first to provide a real-time scale-out demonstration of applying a key insight and philosophy on predicting the interactions between software and microarchitectural resources.

The philosophy that guided our work was based on a paradigm shift in how we understood system behavior. Rather than attempting to create an analytical model of the system, we chose to sample its behavior through perturbation to predict its properties. This approach, while unorthodox, was necessitated by the rising complexity of hardware and the opaque nature of microarchitecture beneath the instruction set architecture interface. Analytical modeling, while appealing from an academic perspective, proved impractical in addressing these challenges.

Looking back, we are heartened to see how our work sparked interest in the broader research community. Bubble-Flux has been cited nearly 500 times in the decade since its publication, inspiring hundreds of subsequent research efforts. These citations demonstrate the relevance and far-reaching impact of our work.

Perhaps most notably, our work illuminated a cross-cutting insight: in high complexity dynamic systems - much like in weather prediction or financial market forecasts - the past can be used to predict the future. We demonstrated that one could manufacture a measurable past by running real-time continuous experiments to create historical information for predicting future behavior. This insight, which we uncovered through our work on Bubble-Flux, has meaningful implications that transcend the realms of computer science and engineering.

The story of Bubble-Flux is a testament to embracing an unorthodox but practical philosophy and method. With Bubble-Flux we not only made a contribution to the field of software and microarchitecture interaction but also offered a broader perspective on understanding and predicting behavior in complex dynamic systems.

References

[1] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: Association for Computing Machinery, 2011, p. 248–259. [Online]. Available: https://doi.org/10.1145/2155620.2155650