PESC Programa de Engenharia de Sistemas e Computação

Luiz Marcio Faria de Aquino Viana <lmarcio@cos.ufrj.br>

[AlunosDPESC] [Sistemas] Fwd: The Importance of End-to-End Thinking in System Design

Felipe Maia Galvao Franca <felipe@cos.ufrj.br> 20 de outubro de 2020 13:23
Para: sistemas <sistemas@cos.ufrj.br>

Sent from my iPhone

Begin forwarded message:

From: SIGARCH <noreply@sigarch.org>
Date: 20 October 2020 10:44:18 GMT-3
To: felipe <felipe@cos.ufrj.br>
Subject: The Importance of End-to-End Thinking in System Design
Reply-To: SIGARCH <noreply@sigarch.org>

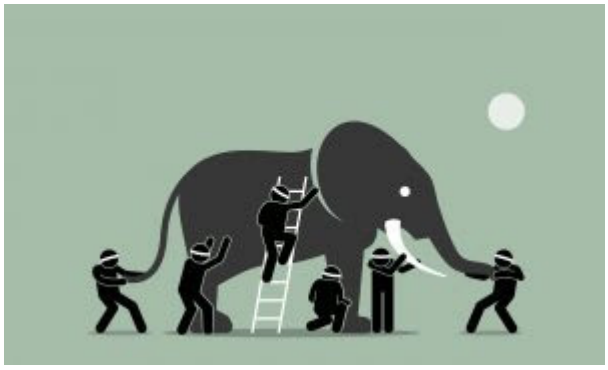
ACM SIGARCH

Computer Architecture Today

Informing the broad computing community about current activities, advances and future directions in computer architecture.

The Importance of End-to-End Thinking in System Design

by Lisa Hsu on October 20, 2020



I once attended a talk by Yoky Matsuoka when I was young – so young, in fact, that I don't remember when or where the talk was. What I do remember is her telling the story of designing a prosthetic hand. She had encountered previous designs where the designers were so bent on creating every mechanical degree of freedom possible, that by the time the joints of the hand and all the cabling required to maneuver the hand was inserted, not only had the hand lost much of the degree of freedom it was supposed to have due to lack of maneuvering

space, but there was also no room for the microcontrollers to actually control it. Thus, the degrees of freedom were physically there, but functionally inoperable. She, on the other hand, was determined to design a maximally functional hand, thus thinking about the design in a more end-to-end manner. This anecdote has stuck with me ever since.

In our field of computer architecture, I occasionally encounter proposals that miss the end-to-end picture. I once saw a presentation from someone proposing a scheme which kept track of history in order to make future predictions. The experiments involved keeping track of 2, 3, or 4 samples of history, and using the average of the stored history to make a prediction. The graphs showed that a history of 3 had the best results, and so the person recommended a history of 3 and dividing the sum of the values by 3 to form the prediction. This is a prime example of lacking end-to-end thinking – the marginal improvement of a history of 3 vs 2 or 4 could not possibly outweigh needing to implement a divider just for this predictor in the total system – the area and power impacts in comparison to a shifter make it clearly the wrong answer – but only if you look end-to-end instead of just at the results in a graph.

The tension here is that as systems get larger and larger, it becomes impossible to consider the entire system end-to-end the whole time. Here at Microsoft in Azure, we operate datacenters in more than 60 regions across the globe, comprising 160+ datacenters connected, arranged into regions, and connected by one of the largest networks on earth. It becomes very difficult to consider what a processor widget here might do in the context of this enormous global system. So what is a processor architect to do? How can one think about the sorts of problems to solve when the end to end system is many, many, many layers of abstraction broader than the proposal at hand?

The following are my tips for evaluating the value of an idea, particularly when the details of the total system may be obscured (i.e., understanding the whole thing is above your pay grade).

- Consider the adjacent steps in the pipeline. Do you know what those adjacent steps are and what they are doing? Does the thing you are proposing make sense in the middle of this minimally expanded scope of consideration?
    - Example: I want to move to a 4-wide execute stage instead of 1-wide, because now you can execute 4x more instructions.
    - But actually: if your fetch stage remains at 1-wide, then going to 4-wide execute won't help at all. This is not thinking end-to-end.
- Consider adjacent layers in the stack – will RTL, circuit, or layout designers tell you to get out of town? Will compiler writers to get out of their office? Does your proposal make sense in the context of the whole development pipeline?
    - Example: I want to propose this new widget and it will mean long wires to communicate across multiple large structures.
    - But actually: if your new design implies long distances across a chip, you will have to either give up cycles, because you have to put flops in the middle of those long wires, or huge driver transistors to push the signals across the wires, costing you power and area, or both.
- Consider the next 1 (or 2) encompassing subsystems, like how it fits into the whole processor or even SoC.
    - Example: My father was an integrated circuit designer-turned-architect, and he told me once that as a young designer he worked on a register file. He made it his mission to make the smallest, most power efficient register file there ever was.
    - But actually: the physical area in the entire design slated for the register file was much larger than what he had come up with, so in fact, he ended up costing power because long wires had to be routed across empty space from the edges of his slot to the ports of his tiny RF.
- Consider the goals of the top-level system. In other words, for many years, computer architects have considered the processor itself as the top-level system, and the goal was to scream through SPEC. We occasionally peeked over at HPC, where the system was a collection of processors and the network that connected them. But now, we are solidly in a world where much of the world's

compute is sitting in a large datacenter, if not in a huge collection of datacenters operated by a hyperscaler like Microsoft, and it's not obvious our community has fully absorbed the implications of this. In this world, a processor is a small participant in this much larger system, and in this system, there are other hugely important metrics besides performance or power (though both are still important).

In this world, where a processor is a participant in a much larger system, my advice is to consider whether a proposal is making the *platform* better. For example:

1. Is it more secure? Does your new context management technique enable workloads from multiple customers share the same platform with the confidence that their workload is secure, from either a data and performance standpoint (or both)?
2. Can it reduce tail latencies? Predictable performance is important in a datacenter environment. It is desirable to avoid scenarios where cloud customers flood a cloud provider with same-price instances, run a characterization test to find out which instances are running on either faster machines or co-located with low-utilization neighbors, and kill the instances that are non-advantageous. Does your new scheduling mechanism enable consistent performance regardless of co-located threads?
3. Does it make virtualization easier? Some enterprise customers want to move their entire workload from datacenters they control (i.e. virtualized), into cloud environments, which mean running a virtualized environment on top of a virtualized environment. Could your new address translation scheme facilitate this?
4. Does it improve network performance (i.e. can it participate better in the broader platform)? Does your new network offload engine reduce the overhead of processing network packets on the host processor?
5. Can it improve debuggability or availability of the platform? Does your new debugging mechanism reduce the reliance on reproducing obscure bugs in the lab that can take millions or hours to encounter? Does your new scheme enable security patches of host OS while minimizing disruption to guest VMs?

When a processor is no longer the top-level system to be designed, then while performance and power remain important, improved value to the larger system in terms of meta-metrics like manageability, security, debuggability, and availability should also receive top billing. To be more general, when designing any system, it behooves the architect to look up, down, left, right, and pop out a few levels to ensure that there is consideration of end-to-end implications. We don't want to be like the story of the blind men with the elephant – feeling the trunk and declaring that elephants are shaped like snakes.

About the Author: *Lisa Hsu is a Principal Engineer at Microsoft in the Azure Compute group, working on strategic initiatives for datacenter deployment.*

Disclaimer: *These posts are written by individual contributors to share their thoughts on the Computer Architecture Today blog for the benefit of the community. Any views or opinions represented in this blog are personal, belong solely to the blog author and do not represent those of ACM SIGARCH or its parent organization, ACM.*

• Email to a friend • View comments • Track comments •

More Recent Articles

by Computer Architecture Today on October 20, 2020

- From Heavy Metal to Irrational Exuberance
- People of Systems & Architecture: Margo I. Seltzer
- Experiences and Lessons from A Virtual Program Committee Meeting
- A Case for Optical Deep Neural Networks
- Architecture Innovation Accelerates Artificial Intelligence

---