An Interview with Indrajit Roy: Toward Self-Correcting Systems

INNOVATION LEADERS

by Bushra Anjum

Editor's Introduction

Indrajit Roy is a staff engineer at Google. He is currently working on peta-scale distributed databases. Previously, he was a principal researcher at HP Labs where he led the development of [Distributed R](#), an open source HP product that brings the benefits of parallelism to data scientists. Roy received his Ph.D. in computer science from UT Austin. He is also an inaugural member of the ACM [Future of Computing Academy](#).

An Interview with Indrajit Roy: Toward Self-Correcting Systems

INNOVATION LEADERS

by Bushra Anjum

Bushra Anjum: What is the next big idea in the field of software engineering that you are dedicating yourself to?

Indrajit Roy: Software engineers are creators. We breathe life into the software. We should build systems that can live without relying on us, the creators. In the industry, we spend a lot of our time tuning and debugging software that we built. We should aim for a future of computing where systems that we create are self-correcting—we are involved in the creation process, but the software itself is responsible for learning and self-managing. Self-correcting software will automatically navigate the changes in the environment such as machines going down in data centers or bugs surfacing in the code. The result will be software that can run for days, months, and years without any human intervention.

We need to invest in research and deployment of self-correcting system software. Consider the unglamorous world of backend software systems; these include distributed databases, resource managers, streaming systems, and so on. An external client, such as a person browsing on the phone, never realizes the presence of these systems, till the backend software fails and causes client-facing issues (e.g., slow payment processing). Even enterprises, like Netflix that are clients of cloud providers, can have service disruptions when a cloud-hosted system goes offline. In this era of 24/7 services, be it Google search or WhatsApp messenger or Amazon cloud services, software failures have to be handled immediately by developers. When a system is down, there can be economic implications for the company. Worse still, there can be adverse implications to clients—what if 911 is unavailable right when there is an emergency?

From the perspective of software developers, we have managed to create complex systems that offer exceptional functionality to clients, yet we rely on dedicated teams of on-call engineers and manual processes to handle failures. As a developer, imagine getting woken up at night to debug performance issues! Won't it be better if software systems were self-correcting? Why do we need to rely on manual intervention to fix systems?

Manual intervention is costly. It is time-consuming and requires deep technical expertise. Instead, we need to move in the direction of building APIs and systems that can self-introspect and take corrective actions.

Today's data center software already incorporates known fault tolerant ideas although to the extent that they are economical. Our systems are multi-homed and run in multiple datacenters. Whenever a new binary is released, it is deployed alongside older versions for hours to days to validate correctness. However, due to economic reasons these systems don't have [NASA's space shuttle like redundancy](#) (e.g., five redundant computers) or use high-end fault-tolerant hardware or incur the code maintenance costs of N-version programming. Given the complex multi-layered software stack we even lack the specification of what is the expected behavior of the software. Therefore, the challenge in creating self-correcting software are these practical deployment constraints because of which we may need to learn the specification of the software, predict when the software may deviate from it, and automatically tune the system. I expect innovative ideas to emerge that use a combination of machine learning, rule-based heuristics, and compiler techniques to make system software self-correcting.

BA: How did this idea of “self-correcting software” emerged?

IR: As I reflect on our education system, a pattern emerges—we learn about fault tolerance techniques, but there is little emphasis on keeping the software running endlessly under practical deployment constraints. I routinely jest amongst friends that the whole purpose of writing software during graduate school was to keep it running long enough to generate that one plot that leads to the one conference paper that helps you graduate.

At Google, I am part of a group that builds and maintains one of the critical databases. We have to develop systems that react to environmental changes and run non-stop. Our clients depend on the system been available all the time. When I compare these expectations to my graduate school days or even my prior experience as part of an industrial research lab, the one thing that jumps out is the focus on keeping the system running no matter what failures occur. Today, this goal is ensured in most companies via dedicated on-call rotations among software developers, i.e., each one of us carries pagers and should be ready to intervene to fix failures manually. Manual intervention prolongs the impact of failures on clients-- humans essentially take more time to react compared to an ideal automated system.

Self-correcting software will require innovation across different computing domains. It will have a significant positive impact on the industry. It also opens up thorny questions around whether this is the beginning of removing humans from the loop completely? Is the end game that there are no software engineers? Maybe, or maybe not. To draw a similarity, having a machine to knead the dough, leaves a pastry chef more time to innovate on her recipes. Similarly, self-correcting systems will let software engineers focus more on creativity instead of the mundane tasks of manually tweaking software in reaction to environmental changes.

BA: The idea of software that can run for days, months, and years without any human intervention is quite exciting. What are the next steps in making this dream a reality?

IR: There are two aspects to advancing the notion of self-correcting systems.

First, we need innovation at the intersection, and possibly integration, of systems and the multitude of learning techniques (e.g., machine learning). Even a simple technique such as applying anomaly detection on the history of distributed operation's latency goes a long way in detecting environmental changes, predicting issues, and then taking corrective actions.

We are adding enough intelligence to our system to automatically handle cases when the workload changes and the system parameters need to be re-tuned. For example, the software may learn that if file system reads are slow in one data center, it should automatically access data from another data center even at the expense of remotely reading data. Similarly, the software may look at history to learn which machines are good candidates to run a task and prefer them to achieve better performance. It may also determine other parameters such as data batching sizes as the workload itself changes.

Second, we need to sensitize the next generation of computer experts with the research challenges in this area. We should encourage interdisciplinary work by students, such as applying machine learning to deployed software. We should also help students to spend time in product development teams, as interns or full-time software developers, to appreciate the need for self-correcting software. Last summer I mentored two undergraduate interns who made improvements to our system to detect anomalous patterns in the deployment environment. It is a tiny contribution in the self-correcting software puzzle but motivated the interns to start thinking about the core challenges in this area and produce artifacts that use ideas from both machine learning and systems research.

A self-correcting system is the start of a conversation. Do you have ideas around which techniques are suitable for implementing self-correcting systems? What avenues should ACM FCA explore to bring together industry and academia around this topic? If you are interested in these questions, reach out to us at the ACM FCA Twitter [account]—[@ACM_FCA](#).

Biography

Bushra Anjum is a software technical lead at Amazon in San Luis Obispo, CA. She has expertise in Agile Software Development for large scale distributed services with special emphasis on scalability and fault tolerance. Originally a Fulbright scholar from Pakistan, Dr. Anjum has international teaching and mentoring experience and has served in academia for over five years before joining the industry. In 2016, she has been selected as an inaugural member of the [ACM Future of Computing Academy](#), a new initiative created by ACM to support and foster the next generation of computing professionals. Dr. Anjum is a keen enthusiast of promoting diversity in the STEM fields and is a mentor and a regular speaker for such. She received her Ph.D. in computer science at the North Carolina State University (NCSU) in 2012 for her doctoral thesis on Bandwidth Allocation under End-to-End Percentile Delay Bounds. She can be found on Twitter [@DrBushraAnjum](#).

DOI: 10.1145/3313830