Evolvable multi-party systems & sophisticated diagnosis tools for the cloud ecosystem

Raja R. Sambasivan

The cloud ecosystem is critical to modern society. We use it when watching movies, buying goods, trading stocks, navigating to destinations, collaborating at work, and even when playing games. This ecosystem is incredibly complex (see Figure 1 for a simplified view). It is comprised of cloud datacenters, eyeball ISPs, and the wide-area network. Cloud datacenters are built and managed by various cloud providers (one provider controls one or more entire datacenters). They run many distributed applications (e.g., shopping applications or distributed-storage applications), which may belong to external tenants (e.g., Target) or internal ones (i.e., cloud providers themselves). Distributed applications may be comprised of 1000s of nodes and any subset of them could be involved in the workflow (i.e., processing) of an application request.

In addition to application nodes, many datacenter stack layers are involved in the processing of application requests. The guest OS layer is controlled by tenants and runs their preferred operating systems. Lower layers are controlled by cloud providers. They include the virtualization layer to multiplex physical resources among tenants, the host OS layer to manage physical computers, the network layer to manage physical switches, and the physical hardware itself. Eyeball ISPs (e.g., Verizon) are Internet service providers that contain tenants’ customers (e.g., online shoppers). The wide-area network (e.g., the Internet) connects cloud datacenters to each other and to eyeball ISPs. It is used to transfer data among applications with nodes in multiple datacenters and to reach customers.

Today’s cloud ecosystem limits innovation because parts of it are not evolvable (i.e., changeable), because evolvable portions are controlled by a single party, and because the difficulty of diagnosing problems in this ecosystem disincentivizes change. As a systems and networking researcher, I am broadly interested in building the multi-party evolvable systems necessary to support innovation within the cloud ecosystem and the diagnosis tools necessary to guarantee their health. Below, I characterize my past and current work in building evolvable systems [1, 3, 8, 11] (marked with parentheses in Figure 1) and in building diagnosis tools [2, 4–7, 9, 10] (marked with angled brackets in Figure 1).

Building evolvable systems: As a PhD student, (A) I helped build Ursa Minor [1, 3, 11], an object-based distributed-storage application whose goal was to support a diverse range of possible current and future datacenter workloads. As a postdoctoral researcher, (B) I implemented a version of of BGP that can facilitate the deployment of the wide range of critical fixes and sophisticated replacements that have been proposed for it. A key result of this research was that small changes to BGP allow for a rich, evolvable public Internet that supports many diverse inter-domain routing protocols.

As a research scientist, (C) I am helping build a public cloud datacenter (called the Mass Open Cloud) that enables innovation by supporting multiple providers as well as tenants. Providers can install a range of infrastructure and services and compete with each other to provide tenants with services. It also aims to make datasets that are typically hidden by cloud providers (e.g., problems logs) available to cloud researchers (e.g., to better support diagnosis research). As part of this effort, (D), I am collaborating with researchers to create an interface that both enables multiple datacenter networks (e.g., ones from different providers) to be installed within datacenters and enables tenants to choose among them.

Building tools for diagnosing systems: My experiences building systems have informed my work on diagnosis. During my PhD, <E> I built one of the earliest workflow-centric tracing infrastructures (also called end-to-end tracing) that strongly supports diagnosis tasks [9]. Such tracing is uniquely suited to informing operational tasks within distributed applications (e.g., diagnosis and resource attribution). With low overhead, it preserves the workflow of individual requests as they are processed within and among the nodes of a distributed application. Workflow-centric tracing works by propagating context (e.g., unique IDs and logical clocks) along with the flow of requests’ execution. This context is stored within records of trace points executed by requests. Workflow traces are created by stitching together trace-point records with related context. Traces are graphs in which vertices represent trace-points. Edges represent causal relationships between trace points and are labeled with inter-trace-point latencies. Today, basic versions of workflow-centric tracing are
being adopted by industry for use in their production systems.

<H> I designed and built Spectroscope [9], one of the earliest automated diagnosis tools that uses workflow-centric traces. Spectroscope helps engineers with their diagnosis efforts by automatically localizing the source of a newly-observed problem from the many distributed-application nodes that may contain the root cause to a few likely candidates. My work on Spectroscope is highly cited (over 160 citations) and was shown to be effective within certain Google infrastructure applications. <G> I created visualizations [7] to effectively present Spectroscope’s results to engineers. My visualizations have influenced recent visualizations in Jaeger, an open-source workflow-centric tracing infrastructure.

<H> I used my experiences with workflow-centric tracing to systematize its design axes [6]. I found that different design choices for these axes dictate a tracing infrastructure’s utility for different operational tasks. I found that design choices that appear innocuous at first can lead to high overheads due to request batching within distributed applications.

As a research scientist, I am advising students on three projects that build on workflow-centric tracing. <F> First, we are extending workflow-centric tracing to capture elements of requests’ workflows within the guest OS layer. This will empower both existing diagnosis tools (e.g., Spectroscope) and the new ones we are creating to diagnose cross-layer problems (e.g., a slow request caused by a bug within the OS). <J> Second, we are building an instrumentation framework that explores the search space of possible instrumentation choices. It will automatically enable the trace points needed to provide insight into a newly-observed problem in running applications. My proposed research on this framework has been just awarded a NSF CSR Small Grant [2] on which I am principal investigator.

<K> Third, we are creating a novel diagnosis abstraction that will mitigate the amount of complexity that engineers (or diagnosis tools) must deal with at any time during diagnosis. We expect that this abstraction will both improve existing diagnosis tools (e.g., Spectroscope) and enable a broad range of new diagnosis-related use cases.

In the following sections, I describe some of my research on building evolvable systems and building diagnosis tools in detail. These efforts were (or are) collaborations with many systems, networking machine learning, and visualization researchers as well as industrial practitioners from Google, LightStep, and Red Hat.

**Enabling evolvability for inter-domain routing**

The Internet’s inter-domain routing infrastructure is a critical piece of its architecture. The routing paths it computes in the control plane connect all of the entities on the Internet (e.g., different datacenters, different ISPs). Today, this infrastructure is provided by a single protocol, BGP, which is extremely limited. It does not provide any quality of service, it advertises only one (potentially poorly performing) best-effort routing path to sources of traffic, it is susceptible to security attacks, and it does not provide any support to help diagnose routing problems. Worst of all, BGP is architecturally rigid—it requires directly neighboring ISPs to use the same inter-domain routing protocol—and thus cannot facilitate the introduction of more advanced inter-domain routing protocols.

BGP’s rigidity disincentivizes ISPs from deploying new routing protocols. ISPs that are interested in deploying a specific new protocol must do so in contiguous groups of isolated islands. These islands cannot discover one another, disseminate new protocols’ information to one another, or use the new protocol to exchange traffic amongst one another. These limitations reduce the benefits of deploying the protocol. Islands can use overlays to forcibly (i.e., without support from BGP) route traffic to islands that have deployed the new protocol. But, the data-plane tunnels that overlays use to force traffic on certain paths can increase the costs of ISPs that are (currently) not interested in deploying the new protocol. This incentivizes them to push back against the protocol’s deployment (e.g., when it is first being standardized).

In this research [8], I identified what features are needed in any inter-domain routing protocol to bootstrap evolution to new inter-domain routing protocols—i.e., facilitate their deployment across non-contiguous ISPs and, if desired, gradually deprecate itself in favor of one of them. By incorporating these evolvability features, these new protocols would themselves
be able to bootstrap evolvability to further new protocols. To identify the features, I systematized what support fourteen recently-proposed inter-domain routing protocols would need to be deployed across (non-contiguous) islands.

The features I identified, pass-through support within routers and multi-protocol support within advertisements, reduce rigidity for two reasons. First, combined, they cleanly separate the information contained in protocols’ connectivity advertisements from that used by their path-selection algorithms. This transforms advertisements into containers that can carry multiple inter-domain routing protocols’ information. Second, multi-protocol advertisements include extra support to allow islands that express their within-island paths in different ways (e.g., path vector and link state) to establish routing paths that include each other. Figure 2 shows a multi-protocol advertisement for a routing path that includes several new protocols from the literature.

To understand the benefits the evolvability features can provide and the difficulty of incorporating them into an existing inter-domain routing protocol, I created a version of BGP (called D-BGP) that includes them. I implemented D-BGP within Quagga, an open-source router. I found that D-BGP is not that far from BGP. It only required ~900 lines of extra code and many (but not all) of D-BGP’s features can be provided by carefully systematizing BGP’s community attributes. I found that D-BGP can support an Internet comprised of many diverse protocols, including critical fixes to D-BGP itself, value-added services, and various other advanced protocols. I also found that D-BGP significantly accelerates the rate at which early adopters see the benefits of adopting specific new protocols at low adoption rates (e.g., at 10–40% adoption) compared to using BGP without data-plane tunnels.

Next steps: The key next step for this research is to explore how D-BGP itself could be deployed. Also, research is needed to contrast the benefits of D-BGP’s approach for enabling evolvability to that of software-defined exchanges (SDXs) and to explore whether a combination of both approaches could yield even greater evolvability benefits.

Automatically localizing performance degradations

A critical step of an application engineer’s diagnosis efforts is problem localization. This challenging task involves identifying which subset of the myriad number of entities (application nodes, code in lower stack layers, ISPs) involved in requests’ workflows are the likely sources of the problem. Once these potential culprit(s) have been identified, engineers can proceed to identify the root cause. To help engineers with this step, my dissertation research involved building and evaluating a tool called Spectroscope that uses workflow-centric traces to automatically localize the sources of performance degradations [7, 9]. Spectroscope’s focus was on the application layer (partially because the workflow-centric traces I used with it captured application-level slices of requests’ workflows).

Spectroscope’s key insight is that performance degradations often manifest as mutations in requests’ workflows. Two types of mutations are possible. Structural mutations involve changes in concurrency, synchronization, or nodes and functions visited by requests. Timing mutations only involve changes in latencies. For a given degradation, exposing the resulting mutations, identifying their previous behavior (called precursors), and showing how mutations and precursors differ localizes the source of the problem. Additional guidance can be provided by ranking mutations and their corresponding precursors by their contribution to the performance change. Spectroscope’s key insight allows it to work without labels indicating whether individual requests are problematic.

Spectroscope takes as input workflow-centric traces from a degraded period of a distributed-applications’ execution and a non-degraded period. It then exploits a variety of domain-specific heuristics, machine-learning algorithms, statistical algorithms, and visualization techniques to achieve its goals (see Figure 3 for a view of Spectroscope’s output). To evaluate Spectroscope, I used it to diagnose eight performance degradations in Ursa Minor and certain Google Infrastructure services. Six of the eight problems were real and previously unsolved. Spectroscope’s results successfully localized the sources of all of the problems.

Next steps: There are four promising next steps, the first three of which I am currently pursuing. They include: 1) Adding support for workflow-centric tracing to the guest OS layer to enable Spectroscope to diagnose cross-layer problems, 2) Creating a just-in-time instrumentation framework to guarantee that the instrumentation needed to sufficiently localize a new problem is present in the system when it is first observed (see next section), 3) Creating a more granular abstraction that Spectroscope can use as input (instead of entire traces) that will allow it to more directly identify mutations and precursors and scale to larger systems, and 4) Creating a new version of Spectroscope whose algorithms are suited for localizing cross-layer problems, using just-in-time instrumentation, and using the more granular abstraction.
Ensuring instrumentation presence via just-in-time instrumentation

When using Spectroscope, I often found that both Ursa Minor and Google lacked instrumentation to provide sufficient visibility into problems. As a result, with Ursa Minor, I often used Spectroscope in time-consuming iterative cycles of localizing the problem using existing instrumentation, adding additional trace points in areas it identified, then re-running Spectroscope. I could not add additional instrumentation to Google applications.

My experiences with Spectroscope reflect a broad truth. It is difficult to know where (e.g., in which nodes), at what granularity (e.g., entire nodes or individual functions), and within which stack layer (e.g., app or guest OS) instrumentation must be added (or enabled) to provide visibility into problems that may occur far in the future. It is also difficult to know what data must be exposed within instrumentation (e.g., queue lengths or function parameters) to provide the needed visibility. Of course, enabling all possible instrumentation would result in unacceptable overheads.

We are currently creating an instrumentation framework (called Pythia) that will automatically enable the instrumentation needed to diagnose a new problem as soon as (or very soon after) it is first observed in a deployed application [2]. Our initial focus is on the application and guest OS layer. Pythia’s key insight is that high performance variation among requests that are expected to perform similarly indicates that there is something important that is unknown about their workflows. This unknown behavior may represent problems (e.g., resource contention) or generally interesting heterogeneity (e.g., uninstrumented code paths that have very different performance profiles) [5].

Localizing the source(s) of the high variation provides insight into where additional instrumentation is needed. Search strategies based on operator knowledge, statistics, or machine learning can then be used to identify what instrumentation is needed to explain it. To diagnose problems that manifest as consistently slow performance (e.g., long queues), a similar procedure that focuses on identifying dominant contributors to request response times can be used.

Using this insight, our current approach for Pythia operates in a continuous cycle of obtaining workflow-centric traces, localizing variation and high response times, and using search strategies to enable or disable instrumentation. Pythia may also choose to do nothing because there are no more unexplained sources of variation or high response times. We are currently building a prototype and exploring causes of high variation within OpenStack. We also plan to apply our Pythia prototype to Ceph. Our prototype controls trace points that have already been added to the application, but it could also use programming-language support to dynamically inject instrumentation into a running system.

Longer-term research plans

My ultimate research goal is to create a rich multi-party cloud ecosystem that supports innovation by being easily evolvable and easy to diagnose. In addition to the next steps identified in previous sections, I plan to pursue the longer-term research listed below to achieve this ambitious goal.

Create mechanisms to correlate and share data relevant to diagnosis among providers, tenants, and ISPs: In the data-plane, workflow-centric tracing enables data from multiple parties to be correlated. Therefore, I plan to instrument additional stack layers using it and explore its applicability to the network (both datacenter and wide area). As part of this effort, I will conduct research to determine if tracing is sufficient for all of the complex behaviors systems can exhibit (e.g., timer interrupts) and propose other primitives if it is not. For the control plane, I plan to explore whether existing provenance mechanisms are sufficient to provide visibility into why decisions were made (e.g., why certain routing or scheduler decisions were made) and proposed alternate mechanisms if not. For both the control and data plane, I will conduct research to 1) determine if information from different parties should be materialized all at once or on-demand, 2) create protocols and frameworks that support these all-at-once and on-demand approaches, and 3) explore how these approaches could be deployed (e.g., via D-BGP).

Identify ways to incentivize (or avoid disincentives for) multiple parties to share information: Despite having mechanisms for sharing data, parties may not wish to share information due to business concerns. To make progress on this important research problem, I plan to identify problem scenarios in which multiple parties must cooperate to avoid poor outcomes—e.g., cases where different parties reacting to a problem independently will exacerbate the problem. The information that must be shared to avoid exacerbating the problem could form a “bare minimum” of data that is always shared. I also plan to explore how other approaches, such as multi-party computation or differential privacy, could help.

Build mechanisms for informed evolution between clouds and the Internet: Such mechanisms would allow clouds and ISPs to fluidly evolve to support one another’s needs. This would spur innovation similar to that only seen within extremely large cloud providers’ private WANs, thus leveling the playing between them and smaller clouds. These mechanisms would also enable ISPs to better support startups operating in new domains (e.g., IoT). As a first step, I plan to explore how our interfaces for allowing datacenter tenants to choose among different datacenter networks (and the routing paths offered by them) could be combined with D-BGP’s and IXPs’ approaches for enabling routing evolvability.
References


