Towards Reliable & Efficient Cloud Systems

My research area lies at the intersection of distributed systems, operating systems, and software engineering, with a particular focus on modern cloud systems. I am interested in efficiently designing, implementing, optimizing, understanding, and improving modern cloud systems.

I develop abstractions, methodology, and tooling to allow developers and researchers to automatically design, implement, optimize, analyse, and improve modern cloud systems.

Cloud systems are developed as loosely-coupled suite of microservices. They are composed of multiple pieces including frameworks, backends, and libraries. These systems are large, diverse, complex and immensely popular in tech companies today. For example, according to a research paper from December 2022, Meta's microservice architecture had 18,500 active services and over 12 million instances; according to a research paper from January 2025, Uber hosts over 6000 microservices. To keep these systems performant and operational, developers must execute three important tasks.

- **Selecting the best design.** For any microservice application, there are many possible designs, each with its own set of performance properties and issues. Predicting the performance implications of a design prior to executing the system is almost impossible due to unpredictable interactions between components. Consequently, the developer must manually try out all possible designs by converting each design into a concrete implementation to measure its performance. This requires a significant amount of manual effort as developers must now build multiple implementations of the systems to find a viable candidate design which is an expensive and time-consuming process.
- **Optimizing for best performance.** Cloud systems operate under dynamic conditions — their workloads change often, they share infrastructure with other applications, components fail randomly, and applications may migrate between different hardware. As a result, the selected design may not be optimal for all possible conditions in which the system operates. Thus, developers need to continuously re-design and re-implement their systems or make sub-optimal generic choices.
- **Understanding problematic behaviors.** Problematic behaviors are unexpected system-wide failures, such as metastability failures, or anomalies that emerge from complex interactions between different components. Analyzing these behaviors to find the root causes of issues is especially laborious as developers must sift through large swathes of data from multiple sources to identify relevant data and then further analyze data to triage issues.

My research tackles the central problem of the extraneous amount of manual effort required to make cloud systems efficient and reliable. To reduce the manual burden on developers and operators, my research focuses on building *human-in-the-loop specification-driven automation tools* that allow developers and researchers to execute system design, implementation, optimization, and analysis tasks.

The key overarching insight underlying my research is to principally decompose these tasks into two parts — (i) a high-level human-provided specification that describes what should be done; and (ii) a tool that automates the execution of specified task. With this approach, developers can specify high-level objectives programmatically, while tools automatically translate these objectives into concrete actions. This approach allows developers and researchers to retain control over goals and tradeoffs, while offloading repetitive and manually extraneous low-level tasks to automated systems.

My research is practical in nature, consisting of fully implemented prototypes. I have built *specification-driven automation tools* that allow developers to automate the design selection and optimization tasks (§1) and automate the capture and analysis of relevant data to enable deep understanding of problematic behaviors ((§2)).

Automating System Design and Implementation Exploration

In the context of system design and implementation, *the goal of my research work is to reduce the manual cost required to efficiently explore the design space of cloud systems.* I focus on two complementary types of exploration - (i) rapid offline prototyping, and (ii) online implementation exploration.

Rapid offline prototyping. Exploration of the microservice application design space is challenging for a multitude of reasons. First, there is an inherent coupling between the design and implementation of the system. To test out the efficacy of the design, one has to fully implement the design. This places a huge manual burden on the developers as to explore the design space they have to implement multiple

variants of the system. Second, given the complexity and variety of the components available, the design space consists of multiple dimensions and large amount of potential design points.

My solution, Blueprint [SOSP23], is a microservice development toolchain designed for rapidly configuring, building, and deploying microservices. Blueprint enables its users to easily mutate the design of an application and generate a fully-functional variant that incorporates their desired changes. The key insight of Blueprint is that the design of a microservice application can be decoupled into (i) the application- level workflow, (ii) the underlying scaffolding components such as replication and autoscaling frameworks, communication libraries, and storage backends , and (iii) the concrete instantiations of those components and their configuration. An application written using Blueprint avoids tightly coupling these concerns. Instead, these design aspects are concisely declared by the user using Blueprint's specifications, namely, the workflow spec and the wiring spec. Blueprint's compiler combines these two specifications to automatically generate the system. Changing any given aspect only requires the developer to revisit its declaration in Blueprint's specifications and not the generated implementation. Moreover, Blueprint eliminates duplicated effort — scaffolding and instantiation logic are implemented once and integrated as Blueprint compiler extensions, to enable Blueprint to automatically change existing Blueprint applications with minimal effort.

Blueprint is an open-source project that is fully operational and actively maintained. It provides a strong foundation for a variety of research directions in building cloud systems. Building on Blueprint, I am developing several systems: Cerulean [AIOps25a; IP1], a tool that generates microservice systems from natural language descriptions; DMAS-Forge [SAA25], a framework for automatically compiling and deploying distributed multi-agent systems; and Palette [APSys25], a framework for creating custom, representative microservice benchmark systems from distributed trace datasets. To further introduce Blueprint to the systems community, I also organized a tutorial on Blueprint at SOSP 2024 [SOSP24Tut].

Online Implementation Exploration. Cloud systems operate in dynamic workload and platform conditions. Specializing implementations of systems to specifics of the workload they serve and platform on which they run often significantly improves performance. In practice, however, specializing systems is difficult due to three compounding challenges: (i) specialization is difficult for developers to implement, as it requires trial-and-error and maintaining multiple versions of key code paths, (ii) specialization fundamentally comes at the cost of generality - a specialized system either performs poorly outside its regime, or completely fails, and (iii) the right combination of specialization choices for different workloads and hardware is almost impossible to predict a priori. This is further exacerbated by the fact that optimal specialization choices are inherently dependent on dynamically changing workloads and platform conditions.

My ongoing project, Iridescent [UR1], addresses these challenges to enable automatic online specialization and optimization of performance critical systems guided by observed overall system performance. The key idea of Iridescent is to specialize performance critical systems at runtime to the exact momentary workload and hardware conditions, without the human developer on the critical path. Iridescent redistributes tasks between ahead of time development and runtime operation. Ahead of deployment, rather than choosing a concrete set of specializations, developers specify the space of possible specializations, along with a search strategy. After deployment, Iridescent's runtime iteratively generates different specialized code versions, choosing the current optimum based on observed overall system performance. With this approach, Iridescent enables both the incremental specialization of existing code-bases with minimal modifications, and in-depth design for specialization for maximizing performance. To accomplish this, Iridescent builds on LLVM to target systems written in typical (non-managed) languages such as C(++) or Rust. Concretely, Iridescent supports the developer in separating the code-base into a specialized, performance critical core, and the remaining generic code. Iridescent then provides the developer with a specialization API to annotate possible specializations in the code thereby specifying the space of possible specializations. Additionally, Iridescent provides hooks to optionally customize JIT code generation in depth. Finally, Iridescent provides the runtime API to control the exploration of different specialized implementations as the system runs.

Enabling Deep System Understanding

Within the context of enabling deep system understanding, my research focuses on two specific problems - (i) capturing relevant traces, (ii) enabling deep visibility into systems.

Capturing Relevant Data. Distributed tracing frameworks capture the end-to-end execution of in-

dividual requests through a distributed system. A trace is a recording of one request that contains the structural and temporal end-to-end flow of a request. This data is then used by developers to identify issues, analyze the behavior of systems, and to do root-cause analysis.

Traditionally, distributed tracing frameworks do not track all incoming requests to avoid performance overheads. Instead, they indiscriminately sample requests to trace when they enter the system. This procedure is called head-based sampling. Head-based sampling is more likely to capture commonly executed paths and misses out on traces from infrequent execution paths. Consequently, it fails to capture problematic relevant edge-case traces because the framework cannot know which requests will be problematic until the request completes. We identified that the crux of this issue is that any sampling technique must make a concrete choice in the trade-off between performance overhead and trace diversity. As a result, we created two different frameworks that support two different trade-off choices.

First, we developed Sifter [SOCC19], a general-purpose framework for making sampling decisions biased towards infrequent and anomalous traces, sacrificing performance in favour of trace diversity. The key challenge was to figure out how to automatically detect anomalous traces in an online setting while be robust to noise and heterogeneity in the trace data. The key idea of Sifter is to use the incoming stream of traces to build an unbiased low- dimensional model that approximates the system's common-case behavior. Sifter uses a simple two-layer neural model that operates on causal sub-paths in traces. Sifter biases selection decisions towards traces that are poorly captured by this model to capturing diverse and anomalous traces. Sifter continuously updates its model with the newly selected traces and automatically adjusts to changes in workload distributions. Sifter is one of the earliest examples of tail-based sampling techniques that are popular in current distributed tracing frameworks.

Next, we built Hindsight [NSDI23], an always-on lightweight distributed tracing framework, which circumvents this trade-off for any edge-case with symptoms that can be programmatically detected, such as high tail latency, errors, and bottlenecked queues. The key insight that enables Hindsight to circumvent this trade-off is that the primary source of performance overhead for distributed tracing is trace ingestion; generating trace data is cheap. Powered by this insight, Hindsight implements a retroactive sampling abstraction: instead of eagerly ingesting and processing traces, Hindsight lazily retrieves trace data only after symptoms of a problem are detected. For requests that were not detected to have a problem, the trace data is simply discarded. Developers using Hindsight receive the exact edge-case traces they desire without undue overhead or dependence on luck.

Enabling Deep Visibility. With the end of Dennard scaling, slowing down of Moore's Law, and the surge of AI workloads, cloud computing is slowly moving towards specialized hardware. Often these Post-Moore systems are built on next-generation hardware, and evaluation in realistic physical testbeds is out of reach. Even when physical testbeds are available, visibility into essential system aspects is a challenge in these heterogeneous hardware systems as system performance depends on sub-μs interactions between hardware and software components. Unfortunately, this visibility cannot be obtained from in-production executions as it would induce large performance overheads. To combat the visibility challenge, we are developing Columbo [IP2], a novel approach to enable in-depth understanding of cloud system behavior at the software and hardware level, with fine-grained visibility. Columbo's key idea is to run cloud systems in detailed full-system simulations, configure the simulators to collect detailed events without affecting the system, and finally assemble these events into end-to-end distributed traces that can be analyzed by existing distributed tracing tools. This allows users to inspect system behavior at a fine-grained level, which is not possible during production.

Future Directions

In my future work, I will use the techniques and abstractions that I have developed as foundational pieces for reliability and efficiency challenges in cloud systems.

Intent Based System Design and Operation. To further reduce the manual effort required for developing cloud systems, I plan to explore using the generative capabilities of LLMs (Large Language Models). To accomplish this, I set forth a long-term vision for achieving holistic automation in a recent vision paper — *intent-based system design and operation* [PACMI25]. We propose *intent* as a new abstraction within the context of system design and operation. Intent encodes the functional and operational requirements of the system at a high-level which can be used to automate design, implementation, and operation of systems. As the first piece of this large vision, we are building Cerulean [AIOps25a; IP1] that combines the generative capabilities of LLMs with the abstractions of Blueprint to automatically

select and generate the input specifications of Blueprint from the high-level user-defined intent.

Efficient Root Cause Analysis. Diagnosing performance issues in a given request requires comparing the performance of the offending request with the aggregate performance of typical requests. Effective and efficient debugging of typical performance issues in distributed systems using distributed trace data faces three challenges: (i) identifying the correct subset of data for diagnosis; (ii) visualizing the data; and (iii) efficient root cause diagnosis from the subset. To combat these challenges, I will build a toolbox of query, visualization, and analysis tooling to better equip users to diagnose issues from distributed trace data. As part of this toolbox, I have already built two tools for comparing the trace of the problematic request with the traces of normal requests. First, I built TraVista [arXiv20], a visual tool that extends the popular single trace Gantt chart visualization with three types of aggregate data - metric, temporal, and structure data, to contextualize the performance of the offending trace compared to all traces. Next, I am building Para*ll*ax [AIOps25b], a compound AI pipeline that enables easy structural and temporal comparison of traces by using the generative abilities of LLMs to first summarize the traces and then performing the comparison on the generated trace summaries. I am currently in the process of integrating both TraVista and Para*ll*ax into Jaeger, a popular widely-used open source distributed tracing analysis tool developed by Uber.

Reliability for Heterogeneous Hardware Cloud Systems. Understanding the end-to-end behavior of Post-Moore systems during deployment requires gaining deep visibility during actual execution of the system. This is necessary for quick detection and mitigation of issues during deployment. For these systems, both performance and energy consumption must be tracked at fine granularity: performance bottlenecks are often caused by subtle interactions between components, and optimizing for energy efficiency depends on attributing consumption precisely for all the various components. To track energy consumption, in my prior work [HotCarbon22], we proposed attributing energy at the request level. However, this is difficult as existing hardware do not support fine-grained measurements and energy consumption is dynamic and changes over time. Moreover, energy is all-encompassing, i.e. every component, small or big, can contribute significantly to the overall energy of the system. We propose that in the short term, we can develop accurate energy consumption models but in the long term we need to make changes into the underlying hardware to obtain actual energy consumption measurements at fine-grained granularity. In a similar vein, for diagnosing performance issues in deployed systems, I plan to build common abstractions for specialized hardware to extract the relevant data for diagnosis.

Lightweight Formal Methods for Emergent Misbehaviors. Complex interactions between components give rise to complex end-to-end misbehaviors that can adversely effect the system's reliability. To combat these misbehaviors, I plan to use techniques from formal methods to help developers find issues in their systems at an early stage and guide developers to safer designs. I have already started exploring this line of work with my ongoing project Bluebell [IP3]. Bluebell combines Blueprint's abstractions with mathematical models of systems to explore the design space of a given system to find potential design points that are more resilient to metastability behaviors in microservice systems. Additionally, I have explored using formal techniques for distributed systems in the past as part of my Master's thesis [FSESRC18; MSc20] where I explored implementation-level model checking techniques for Go-based concurrent systems. Combining these techniques with the abstractions of Blueprint and Iridescent to further study implications of system's design and implementation on its reliability represent an exciting future avenue.

Distributed Experiments on Cloud Systems. There are two main challenges that prevent researchers and practitioners to conduct distributed experiments on cloud systems. First, there is a lack of representative benchmarks and applications that can be used as evaluation targets to evaluate the efficacy of new techniques and infrastructural components. Second, to achieve scientific rigor, these experiments should be easily reproducible in distributed setting. To address the first challenge, I am building Palette [APSys25], which allows users to generate representative microservice systems benchmarks from distributed traces customized for their evaluation needs. Under the hood, Palette uses Blueprint's abstractions to generate the desired system implementations. To address the second challenge, I plan to build abstractions and tooling that allow researchers to programmatically describe their experiments and have them directly link with Palette and Blueprint to automatically execute experiments.

References

[FSESRC18] **Vaastav Anand**. "Dara: hybrid model checking of distributed systems". In: *Student Research Competition, Proceedings of the 2018 26th ACM Joint Meeting on European*

Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM. 2018, pp. 977–979.

[SOCC19] Pedro Las-Casas, Giorgi Papkerashvili, **Vaastav Anand**, and Jonathan Mace. "Sifter: Scalable Sampling for Distributed Traces, without Feature Engineering". In: *Proceedings of the ACM Symposium on Cloud Computing*. 2019, pp. 312–324.

[arXiv20] **Vaastav Anand**, Matheus Stolet, Thomas Davidson, Ivan Beschastnikh, Tamara Munzner, and Jonathan Mace. "Aggregate-driven trace visualizations for performance debugging". In: *arXiv preprint arXiv:2010.13681* (2020).

[MSc20] **Vaastav Anand**. "Dara the explorer : coverage based exploration for model checking of distributed systems in Go". In: (2020).

[HotCarbon22] **Vaastav Anand**, Zhiqiang Xie, Matheus Stolet, Roberta De Viti, Thomas Davidson, Reyhaneh Karimipour, and Jonathan Mace. "The Odd One Out: Energy is not like Other Metrics". In: *HotCarbon* (2022).

[SOSP23] **Vaastav Anand**, Deepak Garg, Antoine Kaufmann, and Jonathan Mace. "Blueprint: A Toolchain for Highly-Reconfigurable Microservice Applications". In: *Proceedings of the 29th Symposium on Operating Systems Principles*. 2023, pp. 482–497.

[NSDI23] Lei Zhang, Zhiqiang Xie, **Vaastav Anand**, Ymir Vigfusson, and Jonathan Mace. "The Benefit of Hindsight: Tracing Edge-Cases in Distributed Systems". In: *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2023), pp. 321–339.

[SOSP24Tut] **Vaastav Anand**, Deepak Garg, Antoine Kaufmann, and Jonathan Mace. *Using Blueprint for accelerating Microservice Research (Tutorial)*. Tutorial at SOSP 2024. 2024.

[AIOps25a] **Vaastav Anand**, Alok Gautam Kumbhare, Celine Irvene, Chetan Bansal, Gagan Somashekar, Jonathan Mace, Pedro Las-Casas, Ricardo Bianchini, and Rodrigo Fonseca. "Automated Service Design with Cerulean (Project Showcase)". In: *2025 IEEE/ACM International Workshop on Cloud Intelligence & AIOps (AIOps)*. IEEE. 2025, pp. 1–3.

[SAA25] Alessandro Cornacchia, **Vaastav Anand**, Muhammad Bilal, Zafar Qazi, and Marco Canini. "DMAS-Forge: A Framework for Transparent Deployment of AI Applications as Distributed Systems". In: *1st Workshop on Systems for Agentic AI* (2025).

[APSys25] **Vaastav Anand**, Matheus Stolet, Jonathan Mace, and Antoine Kaufmann. " Generating Representative Macrobenchmark Microservice Systems from Distributed Traces with Palette". In: *APSys* (2025).

[PACMI25] **Vaastav Anand**, Yichen Li, Alok Kumbhare, Celine Irvine, Chetan Bansal, Gagan Somashekar, Pedro Las-Casas, Jonathan Mace, and Rodrigo Fonseca. "Intent-based System Design and Operation". In: *Practical Adoption Challenges of ML for Systems (PACMI)* (2025).

[AIOps25b] **Vaastav Anand**, Pedro Las-Casas, Rodrigo Fonseca, and Antoine Kaufmann. "Towards using llms for distributed trace comparison". In: *2025 IEEE/ACM International Workshop on Cloud Intelligence & AIOps (AIOps)*. IEEE. 2025, pp. 13–14.

[UR1] **Vaastav Anand**, Deepak Garg, and Antoine Kaufmann. "Iridescent: A Framework Enabling Online System Implementation Specialization". In: *Under Review* (2025).

[IP1] **Vaastav Anand**, Alok Kumbhare, Celine Irvine, Chetan Bansal, Gagan Somashekar, Pedro Las-Casas, Jonathan Mace, Ricardo Bianchini, and Rodrigo Fonseca. "Automated Service Design with Cerulean". In: *In Preparation* (2025).

[IP2] Jakob Goergen, **Vaastav Anand**, Hejing Li, Jialin Li, and Antoine Kaufmann. "Columbo: Low Level End-to-End System Traces through Modular Full-System Simulation". In: *In Preparation* (2025).

[IP3] Zawyar Ur Rehman, **Vaastav Anand**, and Antoine Kaufmann. "Bluebell: A framework for metastability testing of microservice systems". In: *In Preparation* (2025).