Tracing and Debugging Distributed Systems; Programming by Examples

EXPERT-CURATED GUIDES TO THE BEST OF CS RESEARCH

Research for Practice combines the resources of the ACM Digital Library, the largest collection of computer science research in the world, with the expertise of the ACM membership. In every RfP column two experts share a short curated selection of papers on a concentrated, practically oriented topic.

This installment of Research for Practice covers two exciting topics in distributed systems and programming methodology. First, Peter Alvaro takes us on a tour of recent techniques for debugging some of the largest and most complex systems in the world: modern distributed systems and service-oriented architectures. The techniques Peter surveys can shed light on order amid the chaos of distributed call graphs. Second, Sumit Gulwani illustrates how to program without explicitly writing programs, instead synthesizing programs from examples! The techniques Sumit presents allow systems to "learn" a program representation from illustrative examples, allowing nonprogrammer users to create increasingly nontrivial functions such as spreadsheet macros. Both of these selections are well in line with RfP's goal of accessible, practical research; in fact, both contributors have successfully transferred their own research in each area to production, at Netflix and as part of Microsoft Excel. Readers may also find a use case!

As always, our goal in this column is to allow our readers to become experts in the latest topics in computer

science research in a weekend afternoon's worth of reading. To facilitate this process, we have provided open access to the ACM Digital Library for the relevant citations from these selections so you can enjoy these research results in full. Please enjoy! —*Peter Bailis*

OK, BUT WHY? TRACING AND DEBUGGING DISTRIBUTED SYSTEMS

BY PETER ALVARO

Large-scale distributed systems can be a nightmare to debug. Individuallys unlikely events (e.g., a server crashing or a process taking too long to respond to a request) are commonplace at the massive scale at which many Internet enterprises operate. State-of-the-art monitoring systems can help measure the frequency of these anomalies but do little to identify their root causes. Pervasive logging may record events of interest at appropriate granularity, but correlating events across the logs of large numbers of machines is prohibitively difficult.

Distributed tracing systems overcome many of these limitations, making it easier to derive high-level explanations of end-to-end interactions spanning many nodes in distributed computations. But there is no free lunch. Broadly speaking, large-scale tracing systems impose on adopters both an instrumentation burden (the effort that goes into tweaking existing code to add instrumentation points or to propagate metadata, or both) and an overhead burden (the runtime cost of trace

capture and propagation). The collection of papers chosen here illustrates some strategies for ameliorating these burdens, as well as some creative applications for high-level explanations.

Tracing with context propagation

Sigelman, B. H., Barroso, L. S., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., Jaspan, S., Shanbhag, C. 2010. Dapper, a large-scale distributed systems tracing infrastructure; http://research.google.com/pubs/pub36356.html

Dapper represents some of the "early" industrial work on context-based tracing. It minimizes the instrumentation burden by relying on Google's relatively homogeneous infrastructure, in which all code relies on a common RPC (remote procedure call) library, threading library, and so on. It minimizes the overhead burden by selecting only a small sample of requests at ingress and propagating trace metadata alongside requests in order to ensure that if a request is sampled, all of the interactions that contributed to its response are sampled as well.

Dapper's data model (a tree of nested *spans* capturing causal and temporal relationships among services participating in a *call graph*) and basic architecture have become the de facto standard for trace collection in industry. Zipkin (created at Twitter) was the first open-source "clone" of Dapper; Zipkin and its derivatives (including the recently announced Amazon Web Services X-Ray) are in widespread use today.

Mace, J., Roelke, R., Fonseca, R. 2015. Pivot Tracing: dynamic causal monitoring for distributed systems. Proceedings of the 25th Symposium on Operating Systems Principles: 378-393. http://cs.brown.edu/~rfonseca/pubs/mace15pivot.pdf

Dapper was by no means the first system design to advocate in-line context propagation. The idea goes back at least as far as Xtrace, which was pioneered by Rodrigo Fonseca at UC Berkeley. Fonseca (now at Brown University) is still doing impressive work in this space. Pivot Tracing presents the database take on low-overhead dynamic tracing, modeling events as tuples, identifying code locations that represent sources of data, and turning dynamic instrumentation into a query planning and optimization problem. Pivot Tracing reuses Dapper/Xtrace-style context propagation to allow efficient correlation of events according to causality. Query the streams!

Trace inference

Chow, M., Meisner, D., Flinn, J., Peek, D., Wenisch, T. F. 2014. The mystery machine: end-to-end performance analysis of large-scale Internet services. Proceedings of the 11th Usenix Conference on Operating Systems Design and Implementation: 217-231. https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-chow.pdf

What about enterprises that can't (or just don't want to) overcome the instrumentation and overhead burdens of tracing? Could they reconstruct causal relationships after the fact, from unstructured system logs? The

mystery machine describes a system that begins by liberally formulating hypotheses about how events across a distributed system could be correlated (e.g., Is one a cause of the other? Are they mutually exclusive? Do they participate in a pipelined computation?) and then mines logs for evidence that contradicts existing hypotheses (e.g., a log in which two events A and B are concurrent immediately refutes a hypothesis that A and B are mutually exclusive). Over time, the set of hypotheses converges into models of system interactions that can be used to answer many of the same questions.

New frontiers

Alvaro, P., Andrus, K., Sanden, C., Rosenthal, C., Basiri, A., Hochstein, L. 2016. Automating failure testing research at Internet scale. Proceedings of the Seventh ACM Symposium on Cloud Computing: 17-28. https://people.ucsc.edu/~palvaro/socc16.pdf

The *raison d'etre* of the systems just described is understanding the causes of end-to-end latency as perceived by users. Armed with detailed "explanations" of how a large-scale distributed system produces its outcomes, we can do so much more. My research group at UC Santa Cruz has been exploring the use of explanations of "good" or expected system outcomes to drive fault-injection infrastructures in order to root out bugs in ostensibly fault-tolerant code. The basic idea is that if we can explain how a distributed system functions in the failure-free case, and how it provides redundancy

to overcome faults, we can better understand its weaknesses.

This approach, called LDFI (lineage-driven fault injection), originally relied on idealized, fine-grained data provenance to explain distributed executions (see our previous paper, "Lineage-driven Fault Injection," by Peter Alvaro, Joshua Rosen, and Joseph M. Hellerstein, presented at SIGMOD 2015). This more recent paper describes how the LDFI approach was adapted to "snap in" to the microservice architecture at Netflix and to build rich models of system redundancy from Zipkin-style call-graph traces.

Conclusion

Despite the fact that distributed systems are a mature research area in academia and are ubiquitous in industry, the art of debugging distributed systems is still in its infancy. It is clear that conventional debuggers—and along with them, conventional best practices for deriving explanations of computations—must be replaced, but it is too soon to say which approaches will come to dominate. Industry has led in the design and particularly in the popularization of large-scale tracing systems in reaction to a practical need: understanding the causes of user-perceived latency for online services. As these systems become common infrastructure, we will find that this use case is only the tip of the iceberg. The ability to ask and answer rich "why" questions about distributed executions will continue to engender new research that improves the consistency, predictability, and fault tolerance of massive-scale systems.

PROGRAMMING BY EXAMPLES

BY SUMIT GULWANI

PBE (programming by examples) is the task of synthesizing or searching for a program from an underlying program space that satisfies a given set of input-output examples.

A key challenge in PBE is to develop an efficient search algorithm that can discover a program that is consistent with the examples. Various *search techniques* have been developed, including deductive methods, use of constraint (SAT/SMT) solvers, smart heuristics for enumerative search, and stochastic search. Another key challenge in PBE is to deal with the ambiguity in intent specification, since there are many programs that satisfy the given examples but not the user's intent. *Ranking techniques* are used to predict an intended program from within the set of programs consistent with the examples. *Interaction techniques* are used in a refinement loop to converge to an intended program.

PBE has varied applications. It allows end users, 99 percent of whom are nonprogrammers, to create small scripts for automating repetitive tasks from examples. It facilitates software development activities, including program refactoring, superoptimization, and test-driven development. The following sample of recently published work addresses applications from different domains while employing different kinds of search and disambiguation algorithms.

Data manipulation using back-propagation

Gulwani, S. 2011. Automating string processing in spreadsheets using input-output examples. Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming languages: 317-330. https://www.microsoft.com/en-us/research/publication/automating-string-processing-spreadsheets-using-input-output-examples/

The first paper describes a technology for automating string transformations such as converting "FirstName LastName" to "LastName, FirstName". This technology was released as the Flash Fill feature in Microsoft Excel. The paper motivates the design of an expressive DSL (domain-specific language) that is also restricted enough to allow for efficient search. The inspiration came from studying spreadsheet help forums, wherein end users solicited help for string transformations, while describing their intent using examples. The paper describes a domain-specific search algorithm that achieves realtime efficiency, breaking from the previous community tradition of reducing the search problem to querying an off-the-shelf general-purpose constraint solver. The latter, while allowing quicker prototyping, lacks the effectiveness of a custom solution.

The paper also gives first-class treatment to dealing with ambiguity, instead of requiring a larger number of examples, thus improving usability and trust. The search algorithm returns a huge set of programs (represented succinctly) that satisfy the examples, and a ranking function that prefers small programs with few constants is used to guess an intended program.

The success of Flash Fill inspired a wave of interest in both academia and industry for developing PBE technologies for other domains, including number/date transformations, tabular data extraction from log files/web pages/JSON documents, and reformatting tables. With data scientists spending 80 percent of their time transforming and cleaning data to prepare it for analytics, PBE is set to revolutionize this space by enabling easier and faster data manipulation.

Polozov, O., Gulwani, S. 2015. FlashMeta: a framework for inductive program synthesis. Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications: 107-126. https://www.microsoft.com/en-us/research/publication/flashmeta-framework-inductive-program-synthesis/

Developing and maintaining an industrial-quality PBE technology is an intellectual and engineering challenge, requiring one to two person-years. The second paper observes that many PBE algorithms are natural fallouts of a generic meta-algorithm and logical properties of the operators in the underlying DSL. The meta-algorithm is based on back-propagation of example-based constraints over the underlying DSL, reducing the search problem over program expressions to simpler problems over program subexpressions. The meta-algorithm can be implemented once and for all. The operator properties relate to its inverse semantics and can be reused across multiple DSLs.

This allows for construction of a synthesizer generator

that takes a DSL and semantic properties of operators in the DSL and generates a domain-specific synthesizer. With such a generator, PBE technologies become modular and maintainable, facilitating their integration in industrial products. This framework has been used to develop many PBE tools that are deployed in several industrial products, including Microsoft Operations Management Suite, PowerShell 3.0, and the Cortana digital assistant.

Drawings using prodirect manipulation

Hempel, B., Chugh, R. 2016. Semi-automated SVG programming via direct manipulation. Proceedings of the 29th Annual ACM Symposium on User Interface Software and Technology: 379-390. https://arxiv.org/abs/1608.02829

PBE can bring together the complementary strengths of direct GUI (graphic user interface) manipulation (mouse- and menu-based) and programmatic manipulation of digital artifacts such as spreadsheets, images, and animations. While direct manipulation enables easy manipulation of a concrete object, programmatic manipulation allows for much more freedom and reusability (but requires skill). This paper bridges this gap by proposing an elegant combined approach, called prodirect manipulation, that enables creation and modification of programs using GUI-based manipulation of example objects for the domain of SVG (scalable vector graphics).

The user draws shapes, relates their attributes, and groups and edits them using the GUI, and the drawing is kept synchronized with an underlying program. The various

GUI-based actions translate to constraints over the example drawing. Constraint solvers are used to generate candidate modifications to the underlying program so that the resultant program execution generates a drawing satisfying those constraints. Smart heuristics are used to select an intended modification from among the many solutions. A skilled user can edit the resulting program during any step to refine the automatically generated modification or to implement some new functionality.

Superoptimization using enumerative search

Phothilimthana, P. M., Thakur, A., Bodík, R., Dhurjati, D. 2016. Scaling up superoptimization. Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems: 297-310. https://people.eecs.berkeley.edu/~mangpo/www/papers/lens-asplos16.pdf

PBE can be used to solve the general problem of program synthesis from an arbitrary specification, given an oracle that can produce counterexamples where the synthesized artifact does not match the intended behavior. This paper uses this reduction, also referred to as CEGIS (counterexample-guided inductive synthesis), to advance the state of the art in superoptimization, which is the problem of finding an optimal sequence of instructions for a given code fragment.

The heart of the paper is a novel PBE algorithm based on enumerative search that considers programs in the underlying state space in order of increasing size. The algorithm leverages an elegant memoization strategy,

wherein it computes the set of programs of bounded size that satisfy a given collection of examples and incrementally refines this set with more examples in the next iteration. The programs are represented succinctly using their behavior on the example states. The algorithm also leverages a powerful meet-in-the-middle pruning technique based on bidirectional search, where the candidate programs are enumerated forward from input states, as well as backward from output states.

The paper further studies the strengths and weaknesses of different search techniques, including enumerative, stochastic, and solver-based, and shows that a cooperative search that combines these is the best.

The future

PBE can be regarded as a form of machine learning, where the problem is to learn from very few examples and over a rich space of programmatic functions. While past developments in PBE have leveraged logical methods, can recent advances in deep learning push the frontier forward? Another exciting direction to watch out for is development of natural-language-based programming interfaces. Multimodal programming environments that would combine example- and natural-language-based intent specification shall unfold a new era of programming by the masses.

Acknowledgments

Thanks to Ravi Chugh, Phitchaya Mangpo Phothilimthana, and Alex Polozov for providing useful feedback on this article.

Peter Bailis *is an assistant professor of computer science at Stanford University. His research in the Future Data Systems group (futuredata.stanford.edu) focuses on the design and implementation of next-generation data-intensive systems. He received a Ph.D. from UC Berkeley in 2015 and an A.B. from Harvard in 2011, both in computer science.*

Peter Alvaro *is an assistant professor of computer science at UC Santa Cruz, where he leads the Disorderly Labs research group (disorderlylabs.github.io). His research focuses on using data-centric languages and analysis techniques to build and reason about distributed systems in order to make them scalable, predictable, and robust to the failures and nondeterminism endemic to large-scale distribution. Alvaro earned his Ph.D. at UC Berkeley, where he studied with Joseph M. Hellerstein.*

Sumit Gulwani *leads a research and engineering team at Microsoft that develops program synthesis technologies for data wrangling and incorporates them into real products. His programming-by-example work led to the Flash Fill feature in Microsoft Excel used by hundreds of millions of people. Gulwani has coauthored around 50 patent applications, published 110 papers in top-tier conferences/journals across multiple computer science areas, and delivered 30 keynotes/invited talks at various forums. He was awarded the ACM SIGPLAN Robin Milner Young Researcher Award in 2014 for his pioneering contributions to end-user programming and intelligent tutoring systems. He obtained his Ph.D. from UC*

Berkeley and was awarded the ACM SIGPLAN Outstanding Doctoral Dissertation Award. He obtained his bachelor's degree from IIT Kanpur in 2000 and was awarded the President's Gold Medal.

CONTENTS