Sawmill: From Logs to Causal Diagnosis of Large Systems

Markos Markakis  
MIT CSAIL  
markakis@mit.edu

An Bo Chen  
MIT CSAIL  
anbochen@mit.edu

Brit Youngmann  
Technion  
brity@technion.ac.il

Trinity Gao  
MIT CSAIL  
trinityg@mit.edu

Ziyu Zhang  
MIT CSAIL  
sylziyuz@mit.edu

Rana Shahout  
Harvard University  
rana@seas.harvard.edu

Peter Baile Chen  
MIT CSAIL  
peterbc@mit.edu

Chunwei Liu  
MIT CSAIL  
chunwei@mit.edu

Ibrahim Sabek  
University of Southern California  
sabek@usc.edu

Michael Cafarella  
MIT CSAIL  
michjc@csail.mit.edu

ABSTRACT
Causal analysis is an essential lens for understanding complex system dynamics in domains as varied as medicine, economics and law. Computer systems are often similarly complex, but much of the information about them is only available in long, messy, semi-structured log files. This demo presents Sawmill, an open-source system [3] that makes it possible to extract causal conclusions from log files. Sawmill employs methods drawn from the areas of data transformation, cleaning, and extraction in order to transform logs into a representation amenable to causal analysis. It gives log-derived variables human-understandable names and distills the information present in a log file around a user’s chosen causal units (e.g., users or machines), generating appropriate aggregated variables for each causal unit. It then leverages original algorithms to efficiently use this representation for the novel process of Exploration-based Causal Discovery - the task of constructing a sufficient causal model of the system from available data. Users can engage with this process via an interactive interface, ultimately making causal inference possible using off-the-shelf tools. SIGMOD’24 participants will be able to use Sawmill to efficiently answer causal questions about logs. We will guide attendees through the process of quantifying the impact of parameter tuning on query latency using real-world PostgreSQL server logs, before letting them test Sawmill on additional logs with known causal effects but varying difficulty. A companion video for this submission is available online [4].

CCS CONCEPTS
• Software and its engineering → System administration;  • Computing methodologies → Causal reasoning and diagnostics; Natural language generation.

KEYWORDS
Logs, Fault Diagnosis, Causality, LLMs, Causal Discovery

1 INTRODUCTION
Failures are frequent in today’s large, complex computer systems, and diagnosing them in production can be challenging [8]; there is usually not enough time (or often even the access permissions) for debugging techniques like testing [6], formal verification [7] and simulation [11]. Instead, operators have to work backward from failures using observational data collected from the system. Informally, we have heard of operations teams spending tens of hours with tools like Datadog [1], trying to diagnose a problem.

However, operations teams want to go beyond a diagnosis - they want to repair the system by alerting the appropriate engineering team. Moreover, whenever there are multiple ways to fix a problem, they would like to identify the most efficient way to utilize engineering effort. This points to the growing field of causal reasoning [13], which has provided scientists with a common language to express and evaluate hypotheses across diverse domains.

We aim for a general-purpose causal diagnosis system that can address a wide range of software systems problems by helping engineers pose and correctly answer Average Treatment Effect (ATE) queries [13]. Intuitively, the ATE formalizes a "dose-response" model under Pearl’s theory of causality [13] (which we adopt in this work): for example, per "dose" of parallelism (e.g., one more worker), by how much will query latency decrease? Crucially, calculating ATEs correctly from observational data involves adjusting for confounders - common causes of both variables involved in an ATE calculation which could bias the observed effect [13]. For example, the available memory could impact both the chosen degree of parallelization and the query latency directly. Calculating ATEs quantifies the trade-offs involved when several interventions could have some impact, helping engineers pick the most efficient one.

Unfortunately, directly calculating ATEs based on the raw data available to large systems operators is impossible. The raw data usually take the form of collections of logs: semi-structured chronological accounts in text form. Figure 1a presents a snippet of a real-world log from the PostgreSQL dataset that we will use in our demonstration. Past work on log management has led to significant infrastructure, including tools offering extensive coverage of software and hardware components [10]. However, causal inference toolkits require a different representation of the underlying...
a data-driven hypothesis about the cause of an observed failure and retrieve the correctly adjusted associated Average Treatment Effect (ATE). To address Challenge A, we derive human-understandable variable tags by leveraging Large Language Models; for Challenge B, we distill log information around causal units and generate new variables for every causal unit to maximize “usefulness”; while in the face of Challenge C, we propose interfaces to recover a relevant, partial causal model of the system from log-derived data.

SIGMOD ’24 participants will act as analysts using Sawmill to investigate the causes of system failures starting from system logs. This includes inspecting the raw log files, examining the generated dataset, and finding the primary causes of unexpected behavior. We will walk participants through an analysis of real-world PostgreSQL logs collected under different configurations, where max_parallel_workers and work_mem confound each other’s impact on downstream latency on TPC-DS [15] (green edge in Figure 1c). We will then give them the chance to independently use Sawmill to analyze failures in more log datasets of varying difficulty.

2 SAWMILL ARCHITECTURE
Figure 2 summarizes our framework, which transforms logs into a table like Figure 1b and derives a causal graph like Figure 1c, enabling ATE calculations. It starts by Parsing logs and Tagging each variable with a human-understandable tag. The parsed information is then reorganized to satisfy the requirements for causal inference, by Defining Causal Units and Computing Suitable Variables, making Obtaining a Causal Graph and Answering ATE Queries possible. We will now dive deeper into each step.

2.1 From the Log to the Parsed Table

2.1.1 Log Parsing. Logs are first parsed into the parsed table, with one row per log message and one column per log-derived parsed variable. This is performed by calling the function Parse. Users can optionally pass into Parse regular expressions for variables like timestamps. Sawmill then uses Drain [9], an off-the-shelf algorithm, to identify the rest of the parsed variables. To drop uninteresting variables like unique identifiers, we discard any categorical parsed variable with over 0.15% distinct values across V total occurrences, similar to past work [19]. If Drain has mapped semantically different variables to the same parsed variable, the user can identify it in the parsed table and correct it using Sawmill’s SEPARATE function.

2.1.2 Variable Tagging. For variables parsed using a regular expression, the user provides a tag together with the regular expression. Sawmill then automatically assigns a unique human-understandable tag to each of the remaining parsed variables. Sawmill consults three sources in sequence to generate a tag: the tokens
precedes the variable in the corresponding log template; GPT-3.5-Turbo [2], given an example message including the variable and values for the variable from other log messages; and GPT-4 [12], given the same information. If no source produces a tag, or if the produced tag is already assigned to a different variable, Sawmill assigns a unique string of symbols instead.

2.2 From the Parsed Table to the Prepared Table

2.2.1 Defining Causal Units. The parsed table includes one row per log message, but useful ATE queries for troubleshooting are usually about larger units, like sessions or machines. This requires grouping several log messages together as a causal unit, with each group being "one data point" for the ATE query at hand. To define causal units in Sawmill, a user calls SetCausalUnit on the appropriate parsed variable - e.g. sessionID. Not every choice of causal unit is permissible, because of the Stable Unit Treatment Value Assumption (SUTVA) [16]: the outcome of each unit should not depend on the treatments of other units. For example, users may be unsuitable causal units if they share hardware: user A’s work could impact user B’s latency. We defer to the user’s knowledge of the particular system at hand to ensure that this condition is satisfied.

2.2.2 Computing Suitable Variables. There can be a varying number of values for each parsed variable in each causal unit. For example, each may have a different number of query latency readings. To make units comparable, Sawmill replaces such varying-size collections of values with the same aggregate(s) of the values per causal unit. This transformation, triggered using Prepare, yields the prepared table, with one row per causal unit and one column for each prepared variable. Each prepared variable is derived from some parsed variable (its base variable) by using some function (e.g. the mean) to reconcile the values within each causal unit. A number of prepared variables are generated for each parsed variable using different functions. Among prepared variables sharing a base variable, Sawmill then only keeps the one that maximizes empirical entropy, to limit the prepared table size and processing cost. We also let the user optionally impute missing values with a default based on domain knowledge, by passing the value to Prepare.

2.3 From the Prepared Table to ATEs

2.3.1 Obtaining a Causal Graph. To obtain a causal graph, Sawmill combines the data-driven and expert-driven approaches: it helps users incrementally build the relevant part of the causal graph, by making data-driven suggestions that the user evaluates using expert knowledge. We call this approach Exploration-based Causal Discovery. The user begins with an "outcome" variable \(Y\) (e.g. mean query latency) and builds the causal graph around it in two phases:

- **Phase I - Finding a "root cause":** Sawmill helps identify a treatment prepared variable \(T\), which causally affects \(Y\) and is also "actionable". It does so by providing the function ExploreCandidateCauses(\(U\)), which suggests likely causes for a prepared variable \(U\). By iteratively leveraging it, the user can arrive at \(T\), concluding Phase I. To efficiently leverage the user’s attention, Sawmill only presents a pruned list of candidate causes, obtained using LASSO [18]. Because each variable is linearly related to its parents and the set of parents is often expected to be small [5], variables that get assigned a zero coefficient can be safely ignored. The user can inspect each candidate cause \(U’\), ranked by increasing p-value, and decide whether to include \(U’ \rightarrow U\) in the causal graph (Accept(\(U’, U\)) or not (Reject(\(U’, U\)).

- **Phase II - Finding confounders:** The user continues constructing the causal graph to identify a sufficient set of confounders that affect \(ATE(T, Y)\). We measure the user’s progress in recovering the regions of the causal graph that could include confounders by calculating an exploration score: the fraction of accepted/rejected edges among edges that touch at least one node in the current graph. Phase II ends when the exploration score reaches 1. We provide SuggestNextExploration, which returns a prepared variable \(U\) such that calling ExploreCandidateCauses(\(U\)) will yield as many edges relevant to the exploration score calculation as possible, maximizing the user’s decision-making efficiency.

2.3.2 Answering ATE Queries. Having transformed the log data into the prepared table and having obtained the part of the causal graph relevant for their analysis, the user is now ready to calculate the ATE of interest by calling the function GetATE(\(T, Y\)).

3 PROTOTYPE AND DEMONSTRATION

3.1 Prototype System

Our prototype of Sawmill uses ~2800 lines of Python and is available online [3]. For log parsing, we used Drain [9]. For LASSO, we used scikit-learn [14]. We used "backdoor.linear_regression" from DoWhy [17] to implement GetATE.

3.2 Guided Demonstration

In this part of our demo, SIGMOD attendees will act as operators at a database company. Customers are using the company’s database under different configurations, with some experiencing higher mean latency for the same workload. Attendees are interested in correctly determining the impact of the max_parallel_workers parameter on mean latency, by following the four steps below. They have access to the PostgreSQL dataset, which includes logs collected on PostgreSQL 14, into which we loaded TPC-DS [15] for scale factor 1 and sequentially issued the TPC-DS queries, excluding the long-running queries 1, 4, 11 and 74. We ran this workload for different settings of the six key parameters from Figure 1c.

**Step 1: Obtaining the Parsed Table** Users inspect the log and invoke Parse, as shown in Figure 3a. Inspecting the parsed table reveals a parsing miss: distinct variables have been mapped to the long-running queries 1, 4, 11 and 74. We ran this workload for different settings of the six key parameters from Figure 1c.

**Step 2: Obtaining the Prepared Table** Users specify that each session is a causal unit and set SUTVA(\(sessionID\)). They then call Prepare to get the prepared table.

**Step 3: Calculating the ATE** Users ask Sawmill for candidate causes of mean query latency by invoking ExploreCandidateCauses(duration; mean), yielding 11 candidates as shown in Figure 3b. These candidates include 6 variables related to the modified...
3.3 Independent Exploration

SIGMOD attendees will use Sawmill on two more dataset families with known causal effects, under conditions of varying difficulty.

**PROPRIETARY:** This dataset is based on a real log for an HTTP-based application from a large company. It covers a collection of users, a fraction $F$ of which is on a faulty OS version, failing HTTP requests with probability $p_F$. SIGMOD attendees will examine Sawmill’s ability to accurately recover the ATE of the faulty OS version on HTTP failure responses. A lower $F$ and/or a lower $p_F$ reduce the “evidence of faultiness”, making the causal effect harder to discern.

XYZ: This dataset logs the values of $V$ different synthetic variables for each of a collection of “machines”. While the values of most variables are randomly chosen, those of $x$, $y$, and $z$ are not. We have that $\text{ATE}(x,y) = -2$, but this is distorted by confounding by $z$. Moreover, $x$ and $y$ include added Gaussian noise with $\sigma = R$ each time they are logged. SIGMOD attendees will examine Sawmill’s ability to reliably detect and adjust for the confounding introduced by $z$. A higher $V$ and/or a higher $R$ increase the noisiness around the target ATE, making the causal effect harder to discern.

REFERENCES


