CAUSAL-PERF: A Causal Inference Approach for Performance Debugging of Software Systems

Md Shahriar Iqbal MIQBAL@EMAIL.SC.EDU

Mohammad Ali Javidian JAVIDIAN@EMAIL.SC.EDU

Pooyan Jamshidi PJAMSHID@CSE.SC.EDU

University of South Carolina, Columbia, SC, USA 29201

Abstract

Modern software systems are highly configurable and they typically come with an increasing number of configuration options at different layers of the entire computing stack. However, due to the sheer size of the configuration space, complex configuration options' interactions and limited understanding of how a deployed software utilizes system resources at runtime, performance debugging is one of the biggest challenges in software systems deployment. In this paper, we develop CAUSAL-PERF that estimates direct and indirect causal effects of configuration options and events on performance and provides a ranking of effective performance debugging queries. We explore performance debugging of five different software systems to estimate the causal effects using a graphical causal model which can be thought of as maps of dependence structure of probability distribution of system performance given configuration options and events generated at runtime.

Keywords: Causal Effect Estimation; Performance Debugging; Software Systems

1. Introduction

Performance is a key issue for modern software systems deployment. Lately, software systems are becoming highly configurable with increased complexity and a growing number of tunable configuration options such as CPU status, CPU frequency, scheduler policy, etc. that influence system performance e.g., inference time, energy consumption (Sze et al., 2017). One of the fundamental challenges for the practitioners is to debug such systems to optimize the response by detecting performance bottlenecks, essentially due to enormous configuration space, complex interactions among configuration options, costly performance measurements and limited knowledge about how the deployed software utilizes system resources at runtime (Xu et al., 2015).

Even today, the best debugging practice remains to be manual that heavily relies on the knowledge and experience of the practitioners to decompose a performance issue by analyzing system resource utilization to identify bottlenecks using correlational patterns (Wu et al., 2017). For example, Netflix's most recent performance debugging tool, *Atlas*, uses a query based approach to detect performance issues by analyzing metrics such as system resource demand, JVM pressure, errors and timeouts, etc. from different sub-components (Watson et al.). It is not uncommon for a single query to extract thousands of such metrics, subsequently reducing them through correlation with system demand. Such information can be leveraged by a practitioner to identify the root cause of performance degradation. Often practitioners use visualization tools like flamegraph to quickly drill down on where a process is spending excess system resources. Different debugging approaches and profiling tools had been developed over the past decade (Curtsinger, 2016; Yu and Pradel, 2016, 2018). However, relying on correlational patterns without system specific knowledge

for performance debugging can be misleading in this context as it involves expensive performance measurements and a practitioner might completely miss a metric that influences performance.

The central issue again becomes to narrow down to a region due to the relatively high number of possible options that influence performance directly or indirectly via events. Many of these configuration options or events may be completely independent of each other or dependent on options or events that are not considered during debugging. Researchers developed a white-box performance analysis tool to understand the interactions of configuration options and performance that can be used for debugging (Velez et al., 2019). Nonetheless, these method comes with high cost as it involves line-by-line source code analysis and cannot determine the effect of options or events not considered during the experiments. Moreover, one option useful to debug a particular performance objective cannot be rendered useful when used for a different performance objective and one must learn the trade-offs (Kolesnikov et al., 2019). For example, we observed that cache miss patterns cannot be used similarly to debug energy consumption and inference time of an image recognition system. These factors limit the the practitioners ability to debug a software system, effectively and efficiently.

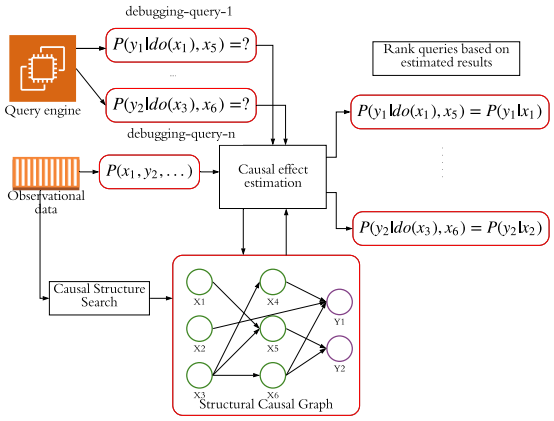

Figure 1: CAUSAL-PERF: Causal graph constructed from the observational data is used to estimate causal effects for debugging queries once identifiable. Queries are ranked based on performance improvement to understand what would have happened if the interventions took place.

In this paper, we address the challenge of effective performance debugging of configurable software systems, using CAUSAL-PERF that constructs a *graphical causal model* from observational performance measurements e.g., energy consumption and inference time. We consider configuration options, such as the number of active CPUs, CPU frequency, scheduling policy, cache statistics, etc. to build our configuration space and track user and kernel events such as context switches, CPU utilization, page faults, etc. observed when the workload is running. The graphical causal model can be queried using different *counterfactual queries* by the query engine as shown in Figure 1. Then we use *do-calculus* causal inference rules to determine the *identifiability* (Pearl, 2000; Huang and Valtorta, 2008; Shpitser and Pearl, 2012) of the *causal effect* of configuration options and events on performance. Once identified, we estimate the causal effects for a counterfactual query using different methods such as inverse probability weighting (IPW), generalized augmented IPW (AIPW), efficient generalized AIPW (eff-AIPW) and gformula. For a series of debugging queries, we rank them based on the performance improvement that would be achieved. Our results show that the ranking of the counterfactual debugging queries using the estimated values obtained from graphical causal model and observational data is consistent with the ranking when the actual intervention is performed using those queries. Thus, CAUSAL-PERF can be used as a **consultant expert system** for prioritizing experiments (perform interventions for the top ranked queries), especially in situations where no clear preferences based on context has been given. Code for reproducing our results and an extended version of the paper are available at `https://github.com/softsys4ai/causal-config-labyrinth`. Our contributions

Table 1: Comparison of total energy consumption in millijoules (mJ) by CPU % utilization for different CPU frequencies in hertz (Hz). Lower energy consumptions are highlighted in bold.

Frequency (Hz) \ Utilization (%)	25	50	75	100	default
652800	**5947**	**6057**	**6123**	**5953**	**6921**
960000	7205	7307	7737	7760	7432
1113600	8442	8941	8775	8434	8678
1881600	9709	10359	10677	10274	13234
2035200 (default)	9818	10426	11168	11123	11257

are the following:
(1) We provide practitioners with a performance debugging method to design experiments for expected performance improvement.
(2) We provide performance engineers a tool to evaluate the effectiveness of the debugging queries.
(3) We perform causal analysis of five different software systems such as image recognition, natural language processing (NLP), speech recognition, SAT Solver and compiler.

2. Definitions and Concepts

We present the necessary concepts to understand the method proposed in the paper.
Causal Graph and Potential Outcome A causal graph $G = (V, E)$ is a directed acylic graph (DAG) which encodes the causal effects between variables, where V is the set of nodes and E is the set of edges. In a causal graph, each node represents a random variable including the treatment, the outcome, other observed and unobserved variables. A directed edge $X \rightarrow Y$ denotes a causal effect of X on Y. The potential outcome of a particular instance Y_T is the outcome that would have been observed if the instance had received treatment T, given the treatment T and outcome Y.
Conditions of Identifiability A causal effect can be estimated only when it is identifiable if conditions of (i) *exchangeability/ignorability* (ii) *consistency* and (iii) *positivity* are hold. The treated ($T = 1$) and untreated ($T = 0$) instances are exchangeable where the assignment of treatment does not depend on the potential outcomes. Consistency is attributed to the unambiguity of a treatment T e.g., that if an instance X receives treatment T by means (route, condition, etc) K, then consistency means that $Y = Y(T, K)$ for any value of K. Positivity refers to not having any instance X on which a treatment T is not possible.
Causal Effect Estimation Causal effect of n instances $\{(X_1, T_1, Y_1), ..., (X_n, T_n, Y_n)\}$ is determined by how the outcome Y is expected to change if a treatment is modified from C to T, which is defined as $E[Y|T] - E[Y|C]$, where T and C denote a treatment and the control. A causal effect is identified iff the *interventional distribution* that explains the causal effect is composed as a function of probability distributions over observed variables. The interventional distribution $P(Y|do(X))$ represents the distribution of the variable Y when we re-generate data by setting the value of variable X to X^*. In the presence of *confounders*, conditions of exchangeability is not satisfied and a causal effect cannot be identified. Confounding is a source of bias that stems from causes of Y which are associated with but not affected by X. A path from X to Y is a confounding path if it ends with an arrow into Y. The variables which intercept confounding paths between X and Y are denoted as confounders.

Table 2: Comparison of total energy consumption in millijoules (mJ) by CPU % utilization for different CPU frequencies in hertz (Hz). Lower energy consumptions are highlighted in bold.

Frequency (Hz) \ Utilization (%)	25	50	75	100	default
652800	**5947**	**6057**	**6123**	**5953**	**6921**
960000	7205	7307	7737	7760	7432
1113600	8442	8941	8775	8434	8678
1881600	9709	10359	10677	10274	13234
2035200 (default)	9818	10426	11168	11123	11257

Counterfactual Query A typical counterfactual query $P(y^*|\widehat{x^*}, x)$ can be expressed in the form: "Given that we have observed $X = x$ in the real world, if X were $\widehat{x^*}$, then what is the probability that Y would have been y^*?"

Estimation Error and Performance Change Estimation error Err of a debugging query is evaluated using the formula: $Err = \frac{|Y-\widehat{Y}|}{Y} \times 100\%$, where Y is the actual outcome when the intervention is performed for a particular debugging query and $\widehat{Y}$ is the outcome estimated by CAUSAL-PERF. We define performance change Δ for a debugging query with $\Delta = \frac{|Y_{T=1}-Y_{T=0}|}{Y_{T=0}} \times 100\%$, where $Y_{T=1}$ is the outcome when a treatment T is applied and $Y_{T=0}$ is the outcome when the treatment is not applied for a particular query.

Configuration The configuration space of a system is defined as the Cartesian product of all options $C = Dom(C_1)... \times Dom(C_d)$, where $Dom(C_i)$ is the set of values to which an option can be set and d is the number of options. A member of this configuration space is known as a configuration.

3. CAUSAL-PERF: A Causal Inference Tool for Debugging

In this section we propose a causal inference approach, called CAUSAL-PERF, to evaluate the performance improvement by counterfactual debugging queries without running new randomized experiments.

3.1 Problem Statement

Performance debugging problem can be formulated as follows: the general goal is to understand which component of a system contributes to what extent to the measured performance. Based on this, it can be decided which components have to be modified to perform a "gradient step" towards the optimal performance. To give an example, a performance engineer may wonder whether the high energy consumption of her image recognition system is caused from the recent code changes made before deployment. As the performance engineer does not have read/write access to the code, she decides to identify what subcomponent of the system such as memory, filesystem etc. is contributing to the increase of energy consumption that would be further controlled for optimization. Let us assume that the performance engineer is interested in evaluating query Q1 given below.

> **Q1:** Is the increased energy consumption due to activities controlled by memory management unit (MMU)?

The engineer needs to provide a deeper look into the components of an MMU the such as virtual memory, page cache, buffere cache etc. to investigate. Let us assume the engineer uses a performance monitoring tool e.g., *perf* and notices unusual swapping activity. She concludes that increased disk I/O resulting from swapping is responsible for performance degradation. Now the performance engineer needs to evaluate the following query Q2 for further drilling down.

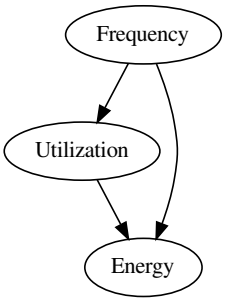

Figure 2: Causal graph obtained using a subsample of performance data from an image recognition system.

Q2: What user/kernel events are responsible for increased swapping activity?

Increase of swapping activity can be either due to one or more of the following: (i) availability of memory resources, (ii) swap usage, or (iii) configuration options that influence swapping behavior such as vm.swappiness, vm.vfs_cache_pressure, vm.vm_page_free_target etc. Hence, the performance engineer needs to perform additional queries and controlled experiments to determine Q2. The additional queries could lead to more experiments making the debugging extremely complex that does not scale well across subsystems. Limited system knowledge on influence of configuration options and their interactions on energy consumption can also confine the performance engineer to focus on a region that does not contribute to performance. In such scnenarios, the performance engineer can utilize CAUSAL-PERF for performance debugging.

CAUSAL-PERF can be used to estimate causal effects of configuration options and events on energy consumption based on performance data alone that the performance engineer has collected in advance without running expensive experiments and determine the top queries that would be useful for improvement.

3.2 Methodology

Performance debugging using CAUSAL-PERF can be performed in the following three steps:

3.2.1 Step 1: Construction of Graphical Causal Model

We discover a directed acyclic graph (DAG) from the observational data. Given a data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ where n is the number of i.i.d. observations of the random vector $X = (X_1, X_2, ..., X_d)$. Let $\mathbb{D}$ be the space of a DAG $G = (V, E)$ on d nodes. Given $\mathbf{X}$, we learn $G \in \mathbb{D}$ for the joint distribution $\mathbb{P}(X)$. To encode the dependency between the variables of X in G we use the NOTEARS algorithm proposed and developed by (Zheng et al.). The three step method from (Zheng et al.) in NOTEARS is based on acyclic structural equation models to learn functions $f = (f_1, f_2, ..., f_d)$ such that $G(f) = G(X)$ using a score-based approach.

NOTEARS converts a combinatorial optimization problem (left) to an equality-constrained program (right) of the following:

$$\begin{aligned} \min_{W \in \mathbb{R}^{d \times d}} \quad & F(W) \\ \text{subject to} \quad & G(W) \in \text{DAGs} \end{aligned} \iff \begin{aligned} \min_{W \in \mathbb{R}^{d \times d}} \quad & F(W) \\ \text{subject to} \quad & h(W) = 0, \end{aligned} \tag{1}$$

where $G(W)$ is the graph with d nodes that is induced by the weighted adjacency matrix W, $F : \mathbb{R}^{d \times d} \to \mathbb{R}$ is a score function and $h : \mathbb{R}^{d \times d} \to \mathbb{R}$ is a smooth function that characterizes acyclic graphs when its levels are set to zero. In the first step, the algorithm uses the augmented Lagrangian method (**?**) to solve the equality-constrained problem that converts a constrained problem (2) into a sequence of unconstrained problems (3) using a quadratic penalty parameter $\rho > 0$, Lagrange multiplier α, augmented Lagrangian L^ρ and dual function $D(\alpha)$.

$$\begin{aligned} \min_{W \in \mathbb{R}^{d \times d}} \quad & F(W) + \frac{\rho}{2}|h(W)|^2 \\ \text{subject to} \quad & h(W) = 0 \end{aligned} \tag{2}$$

$$\begin{aligned} D(\alpha) &= \min_{W \in \mathbb{R}^{d \times d}} L^\rho(W, \alpha) \\ \text{where} \quad L^\rho(W, \alpha) &= F(W) + \frac{\rho}{2}|h(W)|^2 + \alpha h(W), \end{aligned} \tag{3}$$

In the second step, each unconstrained problem in 3 is solved using the proximal quasi-Newton method. Finally, thresholding is performed on the edge weights to obtain consistent estimates of the true graph that rules out cycle-inducing edges. More details on the algorithm can be found in the original paper by (Zheng et al.) that produces the DAG in Figure 2 from the observational data presented in Table 2. In Figure 2, the arrows $Frequency \rightarrow Energy$, $Frequency \rightarrow Utilization$ and $Utilization \rightarrow Energy$ encode the dependence of CPU energy consumption on CPU frequency, CPU frequency on CPU Utilization and CPU utilization on CPU energy consumption, respectively. This DAG can further be utilized to estimate the causal effects of one node over another. Our approach assumes causal sufficiency which means for the DAG in Figure 2 all common causes of $Frequency$, $Utilization$ and $Energy$ are observed: there exists no hidden confounders.

3.2.2 Step 2. Estimation of Causal Effects

The causal effect is defined as difference between the outcome when a treatment T is applied and the outcome when T is not applied. Initially, we discuss how counterfactual queries can be answered to estimate effects using the DAG found in Figure 2 and observational data from Table 2. Later, we elaborate on a necessary precondition i.e., identifiability before estimating causal effects and different causal effect estimation methods adopted by CAUSAL-PERF.

Given sufficient time and sufficient resources, a performance engineer can obtain an (approximate) answer of a debugging query using a controlled experiment. However, instead of carrying out a new experiment, she would like to obtain the answer using data that that had been already collected in the past. CAUSAL-PERF can be utilized to answer counterfactual queries i.e., *"What would have happened if ..."*, *"What would be the effect ... if ..."* etc. questions that can estimate causal effects such as $Frequency \rightarrow Energy$ and $Frequency \rightarrow Utilization \rightarrow Energy$, respectively (Figure 2). Given the example in Table 2 we can estimate the causal effect of following two counterfactual queries:

Example 1 : We are given the following query:

> **$Q_{FREQ\ (default)}$:** "What would be the expected causal effect on CPU energy consumption of changing CPU Frequency from 652800 Hz to 960000 Hz, for default CPU utilization?"

The causal effect of this query $Q_{FREQ\ (default)}$ of a binary treatment T on outcome Y can be formulated as follows:

$$\text{FREQ (default)}_{0\to1}(x) = \mathbb{E}[Y|do(T=1, X=x)] - \mathbb{E}[Y|do(T=0, X=x)] \tag{4}$$

The expectations in (4) can be obtained over the interventional distributions $P(Y|do(T = 1, X = x))$ and $P(Y|do(T = 0, X = x))$. We estimate $Q_{FREQ\ (default)}$ in the following manner:

$$\begin{aligned}
\text{FREQ}_{652800\to960000}(a) \quad &= \mathbb{E}[Energy|do(Frequency = 960000, Utilization = a)] - \\
&\quad \mathbb{E}[Energy|do(Frequency = 652800, Utilization = a)] \\
&= P(Energy = 1|do(Frequency = 960000, Utilization = a)) - \\
&\quad P(Energy = 1|do(Frequency = 652800, Utilization = a)) \\
&= P(Energy = 1|Frequency = 960000, Utilization = a) - \\
&\quad P(Energy = 1|Frequency = 652800, Utilization =)) \\
&= 7432 - 6921 \\
&= 511 \text{ mJ}
\end{aligned} \tag{5}$$

Therefore, for default CPU utilization changing CPU frequency from 652800 Hz to 960000 Hz would increase the energy consumption by 511 mJ.

Example 2 : Let us consider another query Q_{FREQ}, where

> **Q_{FREQ}:** "What would be the expected causal effect on CPU energy consumption of changing CPU Frequency from 1881600 Hz to 960000 Hz?"

The causal effect of this query Q_{FREQ} of a binary treatment T on outcome Y is formulated using the following:

$$\text{FREQ}_{0\to1} = \mathbb{E}[Y|do(T=1)] - \mathbb{E}[Y|do(T=0)] \tag{6}$$

The expectations can be obtained over the interventional distributions $P(Y|do(T = 1))$ and $P(Y|do(T = 0))$. Therefore, to estimate Q_{FREQ} we carry out the following steps:

$$\begin{aligned}
\text{FREQ}_{1881600\to960000} \quad &= \mathbb{E}[Energy|do(Frequency = 960000)] - \mathbb{E}[Energy|do(Frequency = 1881600)] \\
&= \sum_a [P(Energy = 1|Frequency = 960000, Utilization = a) \\
&\quad P(Frequency = 960000|Utilization = a) - \\
&\quad P(Energy = 1|Frequency = 1881600, Utilization = a) \\
&\quad P(Frequency = 1881600|Utilization = a)] \\
&= -3362.4 \text{ mJ}
\end{aligned} \tag{7}$$

This means that by changing CPU frequency from 1881600 Hz to 960000 Hz a performance engineer can reduce the energy consumption by 3362.4 mJ.

Formally, for an outcome that received a treatment ($T = 1$) is denoted by $Y(1)$ and an outcome that did not receive a treatment ($T = 0$) is denoted by $Y(0)$. The observed outcome Y can be written as a linear combination of $Y(1)$ and $Y(0)$ given below:

$$Y = T \times Y(1) + (1 - T) \times Y(0) \tag{8}$$

It is not possible to observe both potential outcomes, $Y(1)$ and $Y(0)$, at the same time. So, we can estimate the average treatment effect (ATE) as the expected difference between the potential outcomes.

$$ATE = E[Y(1) - Y(0)] \tag{9}$$

Under independence assumption (9) can be written as the following:

$$ATE = E[Y(1)] - E[Y(0)] \tag{10}$$

From the observational data we estimate the population average treatment effect $\widehat{ATE}$. We use $\widehat{ATE}$ as the causal estimand that determines the effect of a treatment in this paper.

$$\widehat{ATE} = \frac{1}{n_T} \sum_{i:T_i=1} Y_i - \frac{1}{n_C} \sum_{i:T_i=0} Y_i, \tag{11}$$

where n_T and n_C are the number of samples used for treatment and control (not treated).

It is important to determine which counterfactual queries can be estimated? A query can be estimated if the effect of treatment T on outcome Y can be identified Shpitser and Pearl (2012). $\widehat{ATE}$ of a counterfactual query Q is identified when the following (i) ignorability (ii) consistency and (iii) positivity all hold. We use the ID algorithm (Richardson et al., 2017) to determine the identifiability of the causal effect of treatment T on outcome Y. Once a counterfactual query is unidentified, we remove it from the query engine.

CAUSAL-PERF estimates causal effects (direct and indirect) using different methods such as IPW, AIPW (Glynn and Quinn, 2010), eff-AIPW (Qin et al., 2017) and gformula (Robins and Hernán, 2009; Robins, 1986). CAUSAL-PERF uses (Bhattacharya et al., 2020) implementation that recommends the estimator that can be used for a particular query that achieves the lowest asymptotic variance for a given confidence level ($(1 - \alpha) \times 100\%$, where α is the probability of rejecting the null hypotheses when it is true).

In IPW, the outcome measures are weighted by the inverse of the probability of X that has been assigned to a treatment T using propensity score $p(x)$ given by

$$p(x) = P(T = 1|X = x) \tag{12}$$

For treated samples, we weight their outcome by

$$w(x) = \frac{1}{p(x)}, \tag{13}$$

whereas for control individuals, the weight is:

$$w(x) = \frac{1}{1 - p(x)}, \tag{14}$$

The IPW estimator for ATE, $\widehat{ATE}^{IPW}$ is determined using the following:

$$\widehat{ATW}^{IPW} = \frac{1}{n} \sum_i \left\{ \frac{T_i Y_i}{p\ (X_i)} - \frac{(1 - T_i) Y_i}{1 - p\ (X_i)} \right\} \tag{15}$$

The AIPW estimator for ATE, $\widehat{ATE}^{AIPW}$ is determined using the following:

$$\begin{aligned} \widehat{ATW}^{AIPW} = \frac{1}{n} \sum_i \Bigg\{ & \left[\frac{T_i Y_i}{p\ (X_i)} - \frac{(1 - T_i) Y_i}{1 - p\ (X_i)} \right] - \frac{T_i - p(X_i)}{p(X_i)(1 - p(X_i))} \\ & \left[(1 - p(X_i)) \mathbb{E}(Y_i | T_i = 1, X_i) + p(X_i) \mathbb{E}(Y_i | T_i = 0, X_i) \right] \Bigg\} \end{aligned} \tag{16}$$

Estimation of causal effects using gformula is can be done in three steps from observational data. Firstly, empirical parameters are obtained using $E(Y_a)$. Secondly, counterfactual means $\widehat{E}(Y_a)$ are estimated using $\widehat{E}(Y_a) = \frac{1}{n} \sum_i \widehat{E}(Y | T = a, X = x)$. In the final step, $\widehat{E}(Y_a)$ is used to determine the causal effect by fitting it into the a marginal structure model.

3.2.3 Step 3: Ranking of Counterfactual Debugging Queries

Finally, a number of k queries are ranked based on the estimated performance improvement when all the queries $Q_{j:1 \rightarrow k}$ are estimated. To give an example, for the two estimated counterfactual queries presented in Step 2, Q_{FREQ} would be ranked higher as it predicts higher performance improvement.

Overall, to debug a system using CAUSAL-PERF, a performance engineer would start with the configuration recommended by the top ranked query Q, observe the performance$Y_{T=1}$ with actual intervention on the system and determine estimation error Err. As CAUSAL-PERF provides an estimate of the performance change Δ for a query (extent of the contribution of the configuration change to measured performance) the engineer can determine the usefulness of the query to resolve performance regression. If it resolves the performance regression completely, the engineer does not need to run any further experiments. In case the issue is resolved partially, the engineer should continue with the next query until the issue is resolved or all the ranked queries are considered, whatever happens first. Once the performance engineer observes that the configurations recommended by CAUSAL-PERF cannot resolve the performance regression, she needs to consider additional queries, estimate their effects and repeat the above process.

4. Experimental Setup

We perform experiments in 5 different software systems listed in Table 1. For each software system, we randomly select $|N|$ configurations by varying hardware/OS configuration options and measure inference time, CPU energy consumption and GPU energy consumption. We vary 4 hardware and 4 OS/kernel configuration options values and track total 312 events for different subsystems such as scheduler, memory, block devices etc. using *perf*. We use Nvidia Jetson TX2 and Xavier as our experimentation platforms. With careful observation of the causal graph constructed from the

Table 3: Subject software systems used in performance debugging and control experiments. $|N|$, indicates the number of configurations measured in each system.

Systems	Architecture	Dataset	Hardware	$\|N\|$
Image Recognition	Xception	Imagenet	TX2	1600
NLP	BERT-base	SQuAD 2.0	TX2	2000
Speech Recognition	Deepspeech	Common Voice	Xavier	1200
SAT Solver	3SAT	SATLIB	TX2	2000
Compiler	SaC	SaC-Benchmark	TX2	2000

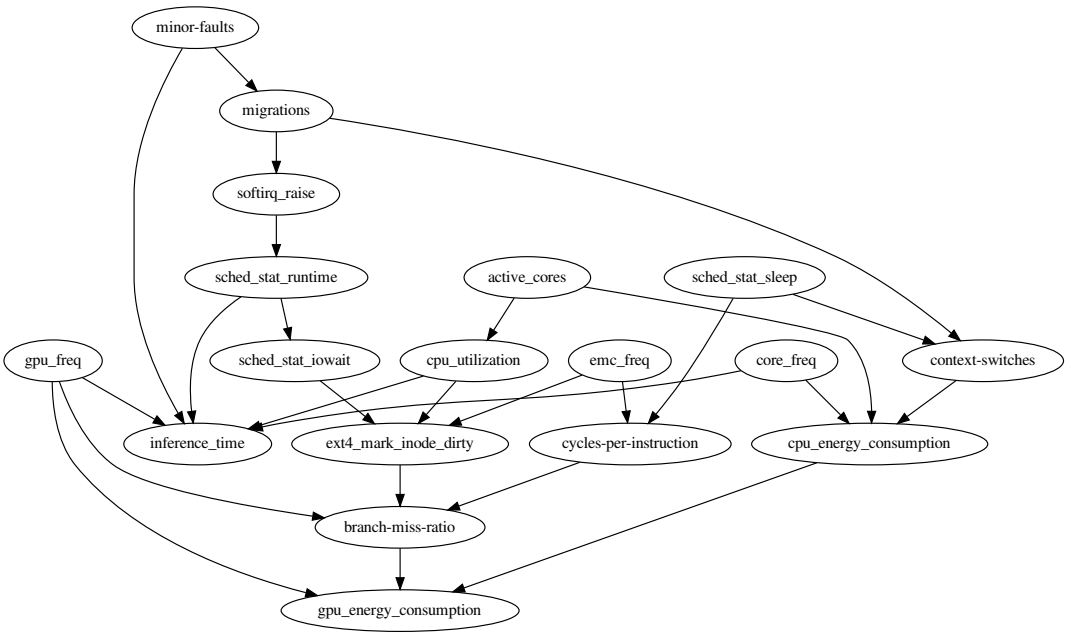

Figure 3: Causal graph obtained using observational data for image recognition system

observational data of a particular system, we build several counterfactual queries to debug different performance objectives. We evalute 20 queries in image recognition system, 19 queries in NLP system and 16 queries for speech recognition, SAT Solver and compiler system, separately. To evaluate the performance estimation of CAUSAL-PERF, we use estimation error Err and performance change Δ. We use Spearman's rank correlation coefficient (Zar, 2005) to validate the ranking of debugging queries recommended by CAUSAL-PERF by comparing ranking using the estimated and actual measurement values for each objective, separately. The rank correlation is useful to evaluate the effectiveness of estimating causal effects of counterfactual queries. Higher rank correlation values for a performance objective in a system would indicate that CAUSAL-PERF can be efficiently utilized to debug that particular performance objective in that system.

Table 4: Estimation errors Err and performance change ΔE_C using CPU energy consumption debugging queries for different software systems. Here, QCPU_{EC1} − $\text{QCPU}_{\text{EC10}}$ refers to CPU energy consumption debugging queries. Queries are arranged with non-increasing order of ΔE_C values for each system. Debugging queries of the same order are not necessarily same between two software systems as the causal graph obtained for each system can be different.

	Image Recognition		NLP		Speech Recognition		SAT Solver		Compiler	
	Err (%)	ΔE_C (%)	Err (%)	ΔE_C (%)	Err (%)	ΔE_C (%)	Err (%)	ΔE_C (%)	Err (%)	ΔE_C (%)
QCPU_{EC1}	6.30	7.12	9.08	5.31	21.47	7.41	7.04	9.03	2.76	3.51
QCPU_{EC2}	4.52	6.20	7.19	3.41	17.53	6.18	5.08	6.08	1.81	2.99
QCPU_{EC3}	4.35	2.13	8.38	1.57	11.47	5.62	6.05	5.58	2.31	1.14
QCPU_{EC4}	5.01	1.14	8.10	0.91	19.44	1.49	4.55	2.01	4.75	0.09
QCPU_{EC5}	4.34	1.11	7.35	-1.98	10.79	1.02	3.22	1.66	3.51	-0.12
QCPU_{EC6}	4.21	0.15	6.59	-2.57	13.70	0.31	6.93	0.81	3.91	-0.75
QCPU_{EC7}	6.54	-0.14	9.17	-3.23	24.40	0.17	0.88	-1.50	1.86	-4.13
QCPU_{EC8}	5.63	-0.16	9.08	-4.13	16.46	-0.51	4.08	-1.90	2.37	-5.38
QCPU_{EC9}	5.25	-1.78	8.54	-4.16	x	x	x	x	x	x
QCPU_{EC0}	5.18	-3.60	x	x	x	x	x	x	x	x

Table 5: Estimation errors Err and performance change ΔE_C using GPU energy consumption debugging queries for different software systems. Here, QGPU_{EC1} − QGPU_{EC3} refers to GPU energy consumption debugging queries. Queries are arranged with non-increasing order of ΔE_C values for each system. Debugging queries of the same order are not necessarily same between two software systems as the causal graph obtained for each system can be completely different.

	Image Recognition		NLP		Speech Recognition		SAT Solver		Compiler	
	Err (%)	ΔE_C (%)	Err (%)	ΔE_C (%)	Err (%)	ΔE_C (%)	Err (%)	ΔE_C (%)	Err (%)	ΔE_C (%)
QGPU_{EC1}	1.05	3.19	2.09	3.82	3.78	0.88	1.04	-1.09	1.91	4.31
QGPU_{EC2}	1.33	0.13	3.67	0.11	6.41	-0.22	0.01	-1.34	2.82	2.56
QGPU_{EC3}	3.13	-1.92	5.41	0.48	5.79	-0.81	3.21	-1.43	0.64	-2.11

To construct causal graphical model from observational data and to determine the causal effect for different counterfactual queries we use causalnex (package repo), causalgraphicalmodels (package repo), and ananke-causal (Bhattacharya et al., 2020; Nabi et al., 2020; Lee and Shpitser, 2020) packages. For causal effect estimation, we set the value of $\alpha = 0.05$. We repeat measurements and estimations 5 times for each query.

Table 6: Estimation errors Err and performance change ΔI_T using inference time debugging queries for different software systems. Here, $Q_{IT1} - Q_{IT7}$ refers to inference time debugging queries. Queries are arranged with non-increasing order of ΔE_C values for each system. Debugging queries of the same order are not necessarily same between two software systems as the causal graph obtained for each system can be different.

	Image Recognition		NLP		Speech Recognition		SAT Solver		Compiler	
	Err (%)	ΔI_T (%)	Err (%)	ΔI_T (%)	Err (%)	ΔI_T (%)	Err (%)	ΔI_T (%)	Err (%)	ΔI_T (%)
Q_{IT1}	5.78	3.45	10.23	0.81	15.17	4.35	2.68	1.31	8.79	2.58
Q_{IT2}	5.13	3.27	6.00	0.79	15.77	2.48	2.76	0.13	6.06	1.98
Q_{IT3}	7.67	1.92	6.67	-0.37	19.70	2.15	2.61	-0.41	6.13	1.19
Q_{IT4}	7.43	1.51	7.53	-0.16	16.19	1.22	3.82	-1.13	6.06	0.02
Q_{IT5}	8.32	-0.16	6.65	-0.78	22.40	0.45	3.50	-1.45	4.97	-1.56
Q_{IT6}	4.76	-1.70	6.35	-0.98	x	x	x	x	x	x
Q_{IT7}	7.10	-1.71	7.08	-1.15	x	x	x	x	x	x

5. Results

The obtained causal graphs shown in Figure 3 is used to find influential configuration options and events on performance. Note, the number of configuration options and events that influence performance are remarkably small (out of the 312 options and events considered in our approach only a few have influence on performance). Our results show that the causal graphs obtained for each of the 5 systems are different (complete causal graph for the remaining 4 systems can be found in the supplementary materials). Therefore, influence of configuration options and events on performance objectives are different across systems. We need to examine each system separately and build counterfactual queries tailored to a particular performance objective for debugging. We use the values estimated by eff-AIPW method to discuss the results as it outperforms other causal effect estimation methods IPW, AIPW and gformula.

We use the results presented in Table 4, 5 and 6 to evaluate the counterfactual queries that we construct by investigating the obtained causal graphs. Table 4 shows the estimation error (Err) and change of CPU energy consumption (ΔE_C) values for each query considered for different systems. We observe that other than speech recognition system (Err per CPU energy consumption query $\equiv 16.90\%$), compiler, SAT Solver, image recognition and NLP systems have relatively lower Err per CPU energy consumption query: 2.91% , 4.72% 5.14% and 8.16%, respectively.

The higher Err in speech recognition system can also be a hardware specific issue (the hardware platform is different than other systems) for which we might need to

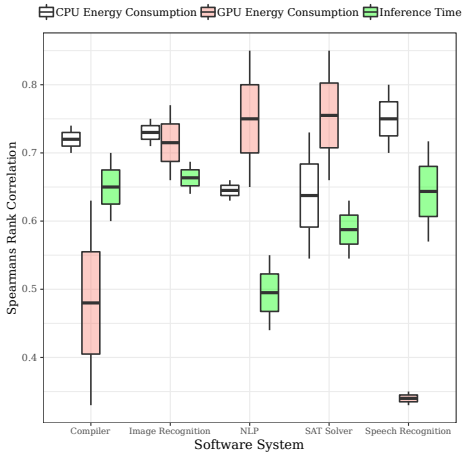

Figure 4: Rank correlation values for energy consumption and inference time debugging queries for different software systems. Higher value indicates better causal effect estimation quality.

consider unobserved confounders more carefully. Table 4 can be also used to determine what debugging queries are effective to decrease CPU energy consumption using the ΔE_C values. To give an example: for image recognition system we can select queries $\text{QCPU}_{\text{EC1}} - \text{QCPU}_{\text{EC6}}$ as they contribute to positive performance change and ignore others. Similarly, for NLP, speech recognition, SAT Solver and compiler systems we can select $\text{QCPU}_{\text{EC1}} - \text{QCPU}_{\text{EC4}}$, $\text{QCPU}_{\text{EC1}} - \text{QCPU}_{\text{EC7}}$, $\text{QCPU}_{\text{EC1}} - \text{QCPU}_{\text{EC6}}$ and $\text{QCPU}_{\text{EC1}} - \text{QCPU}_{\text{EC4}}$ queries, respectively.

Similarly to Table 4, we observe higher Err (per GPU energy consumption query) $\equiv 5.99\%$ and Err (per inference time query$\equiv 17.85\%$ from Table 5 and 6, respectively, for speech recognition system in comparison with other software systems. Furthermore, using ΔE_C and ΔI_T values from Table 5 and 6, we can select effective GPU energy consumption and inference time debugging queries with positive ΔE_C and ΔI_T values, respectively.

We also study the Spearman's rank correlation of the rankings obtained using the estimated measurement values and actual measurements values presented in Figure 4. Our study shows that, ranking of CPU energy consumption debugging queries using estimated values are consistent across all the software systems (close to 0.7). It indicates that causal effect estimation can effectively utilized for debugging CPU energy consumption. Our results also indicate that CAUSAL-PERF can be effectively used to debug GPU energy consumption in image recognition, NLP and SAT Solver and to debug inference time compiler, image recognition and speech recognition systems.

5.1 Discussion

CAUSAL-PERF can be effectively used by practitioners to narrow down the configuration space for performance debugging and to evaluate debugging queries without running expensive experiments. The ranking of debugging queries is useful for practitioners to choose effective queries that predicts performance improvement. This allows the practitioner to scrap queries that do not contribute to performance improvement: saving time time and resources. Predicted performance changes and rankings recommended by CAUSAL-PERF can efficiently be used to design experiments with limiting budgets. For example: specific queries along with number of queries needed to achieve a $x\%$ performance improvement (here, $x \in \mathbb{N}$) can be determined in advance and experiments can be run with more control using CAUSAL-PERF. However, it cannot be guaranteed that the performance improvement predicted by CAUSAL-PERF for successive queries would be cumulative as after performing an actual intervention the causal behavior of the system would change.

6. Related Work

To understand performance behavior across systems different transfer learning methods have been proposed e.g., co-design exploration for embedded systems (Bodin et al., 2016), model transfer across hardware (**?**Valov et al., 2017; **?**; **?**; Jamshidi et al., 2018), and causal analysis of performance modeling of configurable software systems (Javidian et al., 2019). Several optimization methods have been proposed to improve performance objectives by selecting optimal configurations such as search by guessing (Filieri et al., 2015), Bayesian optimization (Jamshidi and Casale), active learning multi-objective optimization algorithm (Zuluaga et al., 2013, 2016), multi-objective optimization algorithm with preference over objectives (Abdolshah et al., 2019), flexible cost aware

multi-objective optimization algorithm (Iqbal et al., 2020) etc. However, in this paper we aim to debug software systems for performance and these methods cannot be used for debugging.

Researchers have developed end-to-end techniques to detect and optimize performance bottlenecks (Yu and Pradel, 2016, 2018). Debugging methods have been proposed for detecting structured workflows using noisy logs (Wu et al., 2017) and anomaly diagnosis (Jia et al.). Performance debugging and analysis methods have also been proposed using data driven approaches (Yu, 2018; Curtsinger, 2016). However, these methods are significantly costly as they require running expensive experiments. Recently, Counterfactual reasoning and causal effect estimation methods have been successfully developed in the context of advertisement recommendation systems (Bottou et al., 2013) and determination of efficient resource sharing strategy for cloud computing (Geiger et al., 2016). Nevertheless, these approaches cannot be applied for the purposes of performance debugging of software systems. Therefore, we develop a causal inference method to performance debug a software system using configuration options and system events.

7. Conclusion

In this paper, we showed how estimation of causal effects using causal models can be utilized for faster and cheaper performance debugging in software systems. To the best of our knowledge, this is the first approach to use causal graphs for performance debugging in software systems using counterfactual queries. For high dimensional configurable systems, bringing the performance of the system, faster, near to its' optimum is a key challenge. To address this, we use observational level performance debugging queries to understand how configuration options and events interact to influence performance. Our approach is particularly useful for softwares deployed in resource constrained IoT and edge devices to fast debug for performance with little experimentation effort. Our method can also be used to get a deeper level understanding of how a particular system exploits a hardware and software during inference. As the state of our work lies, it can be readily used by a practitioner to investigate software systems for performance debugging as an effective starting point. CAUSAL-PERF guides a practitioner to reduce the size of the configuration space, determine what debugging queries can be constructed using the graphical causal model, determine effective debugging queries by removing the ones that do not contribute to performance improvement and design sandbox experiments. As future work, We plan to design a policy based control mechanism that would determine an effective debugging strategy to improve performance.

References

M. Abdolshah, A. Shilton, S. Rana, S. Gupta, and S. Venkatesh. Multi-objective bayesian optimisation with preferences over objectives. *arXiv preprint arXiv:1902.04228*, 2019.

R. Bhattacharya, R. Nabi, and I. Shpitser. Semiparametric inference for causal effects in graphical models with hidden variables. *arXiv preprint arXiv:2003.12659*, 2020.

B. Bodin, L. Nardi, M. Z. Zia, H. Wagstaff, G. Sreekar, M. Emani, J. Mawer, C. Kotselidis, A. Nisbet, M. Lujan, B. Franke, P. H. Kelly, and M. O'Boyle. Integrating algorithmic parameters into benchmarking and design space exploration in 3D scene understanding. In *PACT*. ACM, 2016.

L. Bottou, J. Peters, J. Quiñonero-Candela, D. X. Charles, D. M. Chickering, E. Portugaly, D. Ray, P. Simard, and E. Snelson. Counterfactual reasoning and learning systems: The example of computational advertising. *The Journal of Machine Learning Research*, 14(1):3207–3260, 2013.

C. M. Curtsinger. Effective performance analysis and debugging. 2016.

A. Filieri, H. Hoffmann, and M. Maggio. Automated multi-objective control for self-adaptive software design. In *10th Joint Meeting on Foundations of Software Engineering*, pages 13–24, 2015.

P. Geiger, L. Carata, and B. Schölkopf. Causal models for debugging and control in cloud computing. *arXiv preprint arXiv*, 1603, 2016.

A. N. Glynn and K. M. Quinn. An introduction to the augmented inverse propensity weighted estimator. *Political analysis*, 18(1):36–56, 2010.

Y. Huang and M. Valtorta. On the completeness of an identifiability algorithm for semi-markovian models. *Annals of Mathematics and Artificial Intelligence*, 54(4):363–408, 2008.

M. S. Iqbal, L. Kotthoff, and P. Jamshidi. Transfer learning for performance modeling of deep neural network systems. In *2019 {USENIX} OpML 2019*, pages 43–46.

M. S. Iqbal, J. Su, L. Kotthoff, and P. Jamshidi. Flexibo: Cost-aware multi-objective optimization of deep neural networks. *arXiv*, pages arXiv–2001, 2020.

P. Jamshidi and G. Casale. An uncertainty-aware approach to optimal configuration of stream processing systems. In *MASCOTS 2016*, pages 39–48.

P. Jamshidi, M. Velez, C. Kästner, and N. Siegmund. Learning to sample: exploiting similarities across environments to learn performance models for configurable systems. In *ESEC/FSE*, pages 71–82. ACM, 2018.

M. A. Javidian, P. Jamshidi, and M. Valtorta. Transfer learning for performance modeling of configurable systems: A causal analysis. *arXiv preprint arXiv:1902.10119*, 2019.

T. Jia, P. Chen, L. Yang, Y. Li, F. Meng, and J. Xu. An approach for anomaly diagnosis based on hybrid graph model with logs for distributed services. In *ICWS 2017*, pages 25–32.

S. Kolesnikov, N. Siegmund, C. Kästner, A. Grebhahn, and S. Apel. *Software & Systems Modeling*, 18(3):2265–2283, 2019.

J. J. Lee and I. Shpitser. Identification methods with arbitrary interventional distributions as inputs. *arXiv preprint arXiv:2004.01157*, 2020.

R. Nabi, R. Bhattacharya, and I. Shpitser. Full law identification in graphical models of missing data: Completeness results. *arXiv preprint arXiv:2004.04872*, 2020.

J. Pearl. *Causality: models, reasoning and inference*, volume 29. Springer, 2000.

J. Qin, B. Zhang, and D. H. Leung. Efficient augmented inverse probability weighted estimation in missing data problems. *Journal of Business & Economic Statistics*, 35(1):86–97, 2017.

T. S. Richardson, R. J. Evans, J. M. Robins, and I. Shpitser. Nested markov properties for acyclic directed mixed graphs. *arXiv preprint arXiv:1701.06686*, 2017.

J. Robins. A new approach to causal inference in mortality studies with a sustained exposure period—application to control of the healthy worker survivor effect. *Mathematical modelling*, 7 (9-12):1393–1512, 1986.

J. M. Robins and M. A. Hernán. Estimation of the causal effects of time-varying exposures. *Longitudinal data analysis*, 553:599, 2009.

I. Shpitser and J. Pearl. What counterfactuals can be tested. *arXiv preprint arXiv:1206.5294*, 2012.

N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner. Performance-influence models for highly configurable systems. In *ESEC/FSE*, pages 284–294. ACM, August 2015.

V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

P. Valov, J.-C. Petkovich, J. Guo, S. Fischmeister, and K. Czarnecki. Transferring performance prediction models across different hardware platforms. In *ICPE*, pages 39–50. ACM, 2017.

M. Velez, P. Jamshidi, F. Sattler, N. Siegmund, S. Apel, and C. Kastner. Configcrusher: White-box performance analysis for configurable systems. *arXiv preprint arXiv:1905.02066*, 2019.

C. Watson, S. E. Emmons, and B. Gregg. A Microscope on Microservices. `https://netflixtechblog.com/a-microscope-on-microservices-923b906103f4`.

F. Wu, P. Anchuri, and Z. Li. Structural event detection from log messages. In *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1175–1184, 2017.

T. Xu, L. Jin, X. Fan, Y. Zhou, S. Pasupathy, and R. Talwadker. Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software. In *ESEC/FSE*, pages 307–319, New York, NY, USA, August 2015. ACM.

T. Yu and M. Pradel. Syncprof: Detecting, localizing, and optimizing synchronization bottlenecks. In *25th International Symposium on Software Testing and Analysis*, pages 389–400, 2016.

T. Yu and M. Pradel. Pinpointing and repairing performance bottlenecks in concurrent programs. *Empirical Software Engineering*, 23(5):3034–3071, 2018.

X. Yu. Understanding and debugging complex software systems: A data-driven perspective. 2018.

J. H. Zar. Spearman rank correlation. *Encyclopedia of Biostatistics*, 7, 2005.

X. Zheng, B. Aragam, P. K. Ravikumar, and E. P. Xing. Dags with no tears: Continuous optimization for structure learning. In *NeurIPS 2018*, pages 9472–9483.

M. Zuluaga, G. Sergent, A. Krause, and M. Püschel. Active learning for multi-objective optimization. In *International Conference on Machine Learning*, pages 462–470, 2013.

M. Zuluaga, A. Krause, and M. Püschel. ε-pal: an active learning approach to the multi-objective optimization problem. *The Journal of Machine Learning Research*, 17(1):3619–3650, 2016.