Hound: Causal Learning for Datacenter-scale Straggler Diagnosis

PENGFEI ZHENG, Duke University, USA
BENJAMIN C. LEE, Duke University, USA

Stragglers are exceptionally slow tasks within a job that delay its completion. Stragglers, which are uncommon within a single job, are pervasive in datacenters with many jobs. A large body of research has focused on mitigating datacenter stragglers, but relatively little research has focused on systematically and rigorously identifying their root causes. We present Hound, a statistical machine learning framework that infers the causes of stragglers from traces of datacenter-scale jobs. Hound is designed to achieve several objectives: datacenter-scale diagnosis, interpretable models, unbiased inference, and computational efficiency. We demonstrate Hound's capabilities for a production trace from Google's warehouse-scale datacenters and two Spark traces from Amazon EC2 clusters.

CCS Concepts: • **General and reference** → **Measurement**; **Performance**; • **Computing methodologies** → **Causal reasoning and diagnostics**; **Topic modeling**; • **Software and its engineering** → **Cloud computing**;

Additional Key Words and Phrases: datacenter; distributed system; performance modeling; performance diagnosis; machine learning; causal reasoning; topic modeling

ACM Reference Format:
Pengfei Zheng and Benjamin C. Lee. 2018. Hound: Causal Learning for Datacenter-scale Straggler Diagnosis. *Proc. ACM Meas. Anal. Comput. Syst.* 2, 1, Article 17 (March 2018), 36 pages. https://doi.org/10.1145/3179420

1 INTRODUCTION

Stragglers threaten performance from datacenter-scale parallelism. Datacenters split a computational job into many tasks, execute them in parallel on many machines, and aggregate results when the last task completes.[1] Stragglers are exceptionally slow tasks within a job that significantly delay its completion. Unfortunately, stragglers' effects increase with the number of tasks and scale of the system. In a Google datacenter [49], we find that stragglers extend completion time in 20% of jobs by more than 1.5×.

Prior studies mitigate stragglers with speculative re-execution and scheduling. Re-execution detects and replicates stragglers for computation on another machine [24, 72]. Scheduling avoids machines predicted to perform poorly for a task [52, 70]. These mechanisms implicitly assume

[1] A job contains one or more tasks that execute a single program on multiple data in parallel. When frameworks, such as Apache Spark, organize tasks into stages, each stage corresponds to a job.

Authors' addresses: Pengfei Zheng, Duke University, Department of Computer Science, USA, pfzheng@cs.duke.edu; Benjamin C. Lee, Duke University, Department of Electrical and Computer Engineering, USA, benjamin.c.lee@duke.edu.

2476-1249/2018/3-ART17 $15.00
https://doi.org/10.1145/3179420

stragglers arise from adverse system conditions that can be resolved by assigning tasks to appropriate machines. Despite these efforts, stragglers persist and extend average job completion time by 47% and 29% in Facebook and Microsoft's datacenters [1].

Existing mitigation strategies are incomplete solutions because they address symptoms rather than diagnose causes. For example, stragglers often arise from the skewed distribution of data across tasks. When some tasks are overloaded with work [33], speculative re-execution or rescheduling consumes system resources without improving performance. Although profilers support straggler diagnosis by producing massive datacenter traces, deriving causal explanations from complex datasets is difficult. Existing diagnosis procedures rely heavily on human expertise in systems and application of best practice, which are laborious and fail to scale [1, 2, 23]. Thus, understanding stragglers' causes is a prerequisite for efficient countermeasures and large datasets motivate methods for rigorous causal analysis.

We present a statistical machine learning framework – Hound – that infers stragglers' causes with three techniques. First, Hound models task latency using system conditions, such as hardware activity, resource utilization, and scheduling events, in order to reveal causes of stragglers for each job. Second, Hound discovers recurring, interpretable causes across jobs by constructing topic models that treat jobs' latency models as documents and their fitted parameters as words. Topics reveal sophisticated causes that explain stragglers at datacenter scale. Finally, Hound guards against false conclusions by constructing an ensemble of predictive, dependence, and causal models and reporting only causes found by multiple models.

We demonstrate Hound on two representative systems. The first is a month-long trace from a production Google datacenter with 12K machines running 650K jobs and 25M tasks [49]. The second is a pair of traces from Amazon EC2 clusters running Spark data analytics [46]. Hound produces interpretable topics that describe stragglers' causes. Each topic is a set of system conditions that combine to explain a major source of stragglers that affects many jobs. Results show that Hound's inferred causes consistent with those from expert analysis.

2 SYSTEM OBJECTIVES

We architect Hound for four desiderata: **datacenter-scale diagnosis** to automatically identify recurring causes of stragglers across many jobs; **interpretable models** to concisely reveal sophisticated causes and domain insight; **unbiased inference** to reduce the risk of false explanations when model assumptions are invalid; **computational efficiency** with methods that are parallelizable and have polynomial complexity.

2.1 Datacenter-scale Diagnosis

Datacenter operators benefit from a broad view of stragglers. A datacenter-scale perspective reveals recurring problems that cause stragglers in many jobs instead of minor problems that impact just a few. Hound diagnoses stragglers at datacenter-scale with a two-stage approach. For each job, it infers models of latency and characterizes stragglers' causes. Then it synthesizes patterns across jobs and their models.

Straggler diagnosis requires a separate model for each job rather than a single model for all jobs. Jobs are heterogeneous and completion time varies significantly due to differences in input data, system parameters, and task parallelism. A single model across jobs cannot account for a straggler's slowness relative to other tasks within its job. It cannot differentiate a slow task in an inherently fast job from a slow task in an inherently slow job—the former is a straggler while the latter is a nominal task. Moreover, a single model provides only one causal explanation for thousands of unique jobs when multiple, diverse causes exist and tailored explanations are required.

Datacenter-scale diagnosis requires extracting domain insight from recurring patterns across jobs and their models. Hound use meta learning to identify datacenter-scale patterns [15]. The base learner for each job constructs latency models and reveals system conditions associated with poor task performance. From base learners' models, the meta learner discovers recurring, straggler-inducing conditions shared by many jobs.

Our choice of learners is deliberate and serves several objectives. We select base learners to produce succinct models that can be trained without manual intervention and to support subsequent meta learning. We exclude Bayesian networks [18, 19, 54, 61, 73] and decision / regression trees [6, 12, 56, 62, 63, 69], popular methods for analyzing system performance, because learning patterns from heteromorphic graphs and trees is difficult. We select meta learners to extract semantic structure from jobs' models and to enhance interpretability. Topic modeling infers themes from recurring clusters of words that appear in many documents. Hound constructs topic models by treating base learners' models as documents, system conditions as words, and causal explanations for stragglers as topics.

2.2 Interpretable Models

Widely used models are difficult to interpret. Some methods are prone to over-fitting and preclude broader interpretation. Regression trees often produce large models from small datasets [40]. Such models generate accurate predictions, but cannot produce generalizable insight. Other methods require domain expertise. Lasso regression selects features associated with stragglers and discards the rest [10, 11, 38]. Translating these features into causes requires system expertise.

Hound ensures interpretability with topic models. Latent Dirichlet allocation identifies features that often appear together in jobs' latency models. A cluster of features correspond to a topic, which reveals clear and concise system conditions that are associated with atypically large latencies. For example, the topic [CACHE_MISS(+), CPI(+)] indicates some stragglers arise from poor cache behavior, measured in terms of the number of cache misses and average cycles per instruction. When jobs suffer from diverse causes of stragglers, Hound reports a mix of relevant topics and assesses their relative contributions to system performance.

2.3 Unbiased Inference

A statistical model makes assumptions that prevent it from performing well on all datasets. The No-Free-Lunch theorem states that a learner pays for performance on some datasets with degraded performance on others [68]. Certain models capture some system behaviors but not others [63]. For example, regression assumes little collinearity between predictors. If collinearity exists, fitted models infer erroneous associations [25]. Bayesian networks assume a prior distribution (*e.g.*, Dirichlet) when inferring network structure. Whether the prior is appropriate for the dataset determines inferred network's quality [59].

Relying on one method is risky when analyzing large, heterogeneous datasets such as datacenter traces. Hound uses ensemble learning to reduce risk of biased conclusions. An ensemble combines responses from multiple, independent learners with a majority rule that amplifies correct responses and avoids erroneous ones [13]. An ensemble is robust when its learners are diverse because, given a dataset, many assumptions hold even when some fail. We design an ensemble that combines distinct but complementary methods for diagnosing stragglers.

2.4 Computational Efficiency

Statistical inference can be computationally expensive and even intractable. For example, inferring Bayesian networks requires finding directed acylic graphs that optimally represent the structure of conditional dependencies within the training data, an NP-hard problem [44]. Even after the

network is built, responding to exact queries is intractable [20]. The computational complexity of constructing a model for straggler diagnosis depends on three factors: number of profiled metrics per task, number of tasks per job, and number of jobs. Datacenter traces can include millions of jobs and tasks, each with tens of profiles. Hound relies on learning methods that have polynomial complexity and are amenable to parallelization in a distributed system.

3 THE HOUND FRAMEWORK

Hound infers predictive, dependent, and causal relationships for task latency from job profiles. Moreover, it reveals and assigns relevant causes to each job's stragglers. Hound provides these capabilities by extending state-of-the-art methods to improve inference for datacenter profiles. Figure 1 shows how inputs translate into outputs in three stages. First, base learning captures relationships between latency and system conditions. Second, meta learning reveals recurring topics. Third, ensemble learning integrates results from disparate types of models.

Inputs – Profiles. Illustrated in Figure 2, Hound requires profiles from jobs that exhibit long-tailed latency distributions. Datacenters may collect these profiles with a unified facility, such as Google-wide Profiling [51] or CPI^2 [74], or combine existing facilities for continuous profiling and distributed file systems [3, 28, 29]. Profiles collected throughout the datacenter and across time comprise a dataset. Hound uses the dataset to infer models that take latency as the dependent variable and system conditions, such as resource usage and scheduling events, as independent variables.

Outputs - Causal Topics. Hound's topics are concise causal explanations for stragglers. Each topic is a set of abnormal profile metrics associated with stragglers across many jobs. Suffixes (-) or (+) indicate whether metric's values are significantly lower or higher for stragglers than those for normal tasks. Hound identifies relevant topics for each job and estimates each topic's significance across the datacenter's tasks and jobs. Administrators can use these outputs to easily identify significant causes and triage system anomalies.

Illustrated in Figure 3a, Hound reports a small number of causal topics. Topic E_1 attributes stragglers to lower average and peak processor utilization (APU, PPU). Topic E_3 attributes stragglers to higher garbage collection frequency and duration (GCF, GCD). In Figure 3b, Hound assigns relevant causes to each job, using weighted mixes when a job's stragglers have multiple causes. Hound attributes job J_1's stragglers to a mix of low processor utilization (E_1) and garbage collector interference (E_3). Processor utilization is dominant and weighted 0.83.

3.1 Base Learning

Base learners infer relationships between task latency and profiled metrics for each job. A base learner produces a causality profile C, a vector in $[-1, 1]^P$ where C_i is the effect of metric i on latency and P is the number of metrics. Vector elements are absolute-sum-to-one such that $\sum_{i=1}^{P} |C_i| = 1$. Because relying on a single learner could induce bias and produce false causes, Hound uses an ensemble of heterogeneous learners to discover predictive, dependence, and causal relationships.

- **Predictive (PR).** Model effects of independent variables on the dependent variable, minimizing differences between data and model outputs. Variables with larger (smaller) effects have higher (lower) predictive power.
- **Dependence (DP).** Model association between independent and dependent variables with probabilistic foundations.
- **Causal (CA).** Model cause and effect with matching methods, which compare data that differ only in the suspected cause.

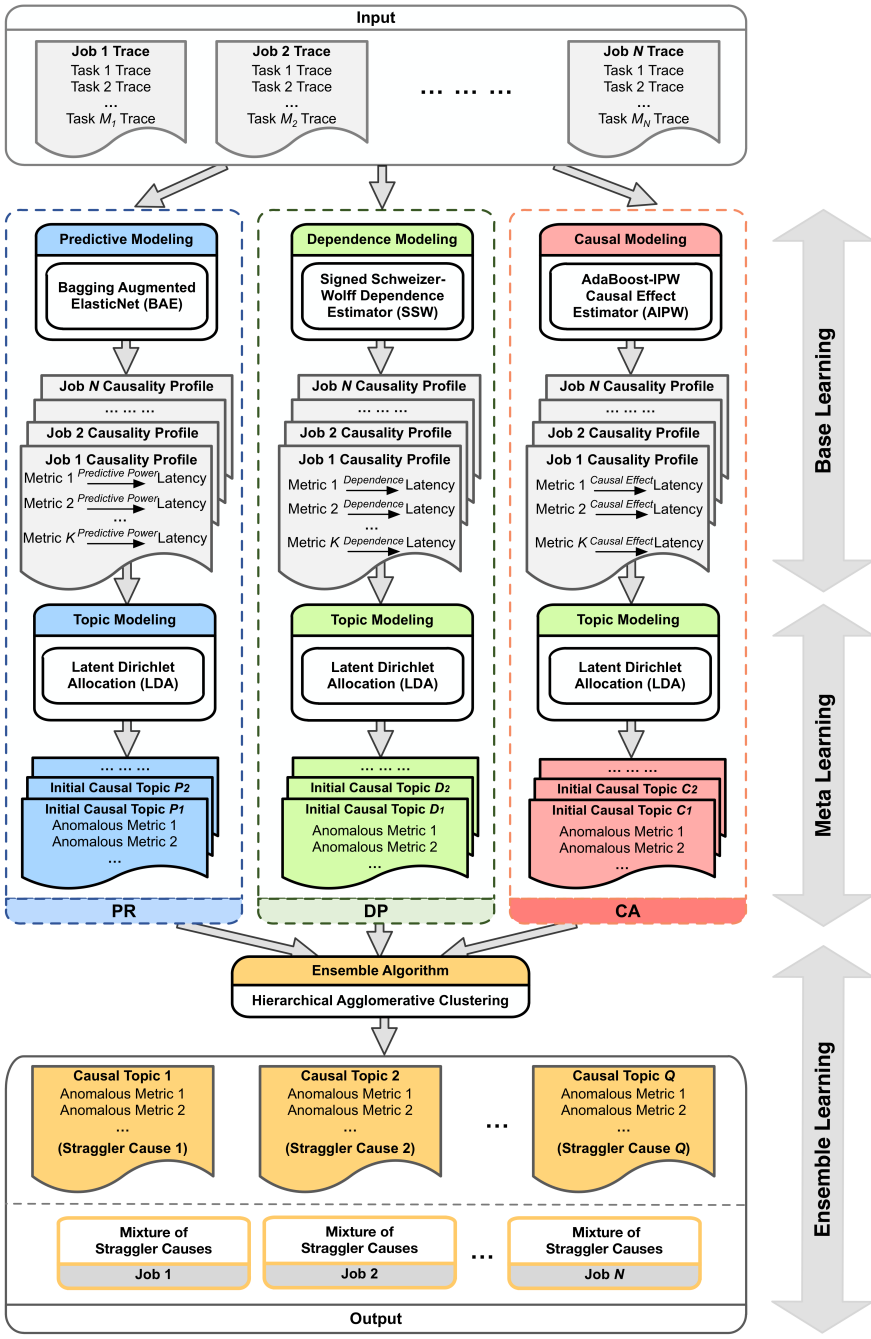
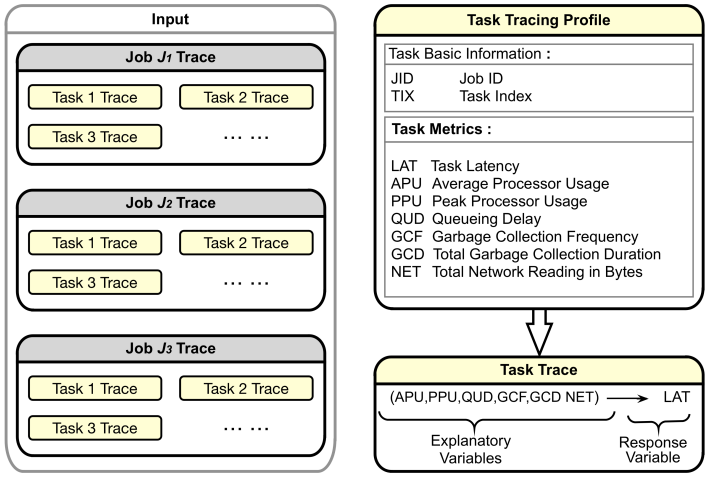


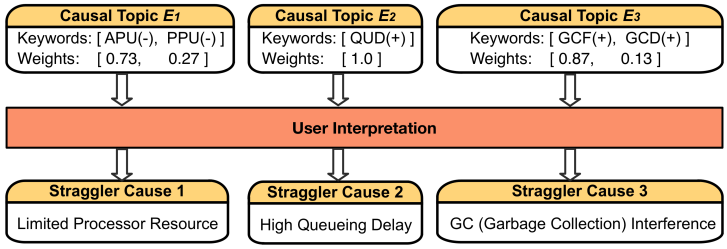
Fig. 1. The Hound Framework

Figure 4 illustrates base learning. The learner infers predictive models with regularized regression methods such as ElasticNet. It then produces a causality profile from fitted and re-scaled regression

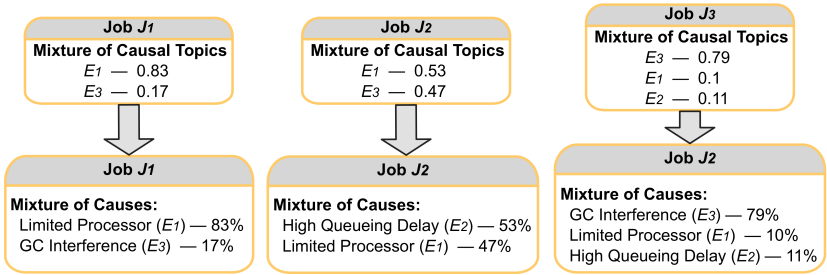


(a) Jobs and tasks (b) Task profile and trace

Fig. 2. Example - Hound Inputs



(a) Inferred causes of stragglers across jobs



(b) Assigned causes of stragglers for individual jobs

Fig. 3. Example - Hound Outputs

coefficients. The causality profile reveals each metric's statistical significance when predicting latency. For Job J_2, low processor utilization (APU, PPU) and high queueing delay (QUD) predict high task latency. Garbage collection and network communication have no effect.

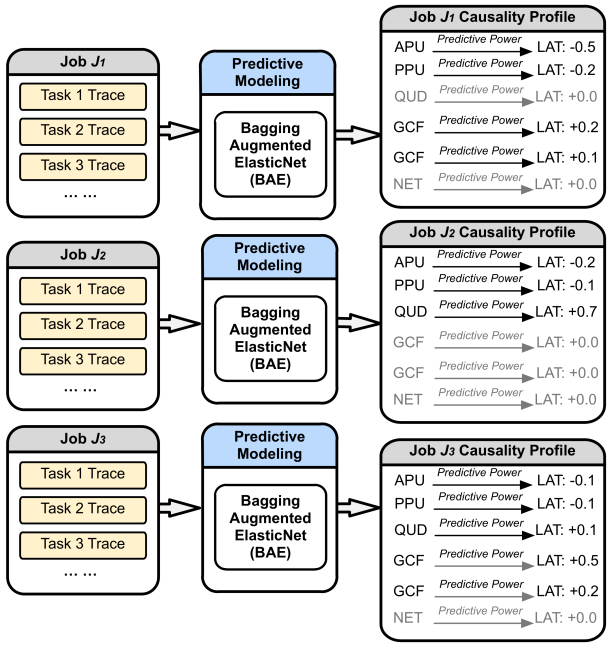


Fig. 4. Example - Hound Base Learning

Predictive Modeling (PR). Linear regression supports the creation of causality profiles and subsequent meta learning. Hound's PR learner constructs regression models with Bagging Augmented ElasticNet (BAE). ElasticNet is a regularization method that automatically selects significant metrics [77]. Bagging is a machine learning method that improves model accuracy and stability. Hound combines these methods to fit latency models using profiled metrics. It encodes resulting regression models as vectors of coefficients to quantify metrics' predictive power and summarize potential causes of stragglers.

Regularization methods enable robust regression by addressing collinearity. Collinearity, which is typical in computer systems, arises when correlations between variables distort estimates of regression coefficients [25]. Lasso, a popular regularization method in systems research [11], mitigates collinearity by randomly including only one variable from a group of correlated variables [64]. However, randomly excluding variables may harm diagnostic power. Lasso may, for example, include cycles per instruction (CPI) and exclude correlated hardware events such as cache misses or branch mispredictions. A model that includes only CPI would fail to identify cache behavior as a more likely and direct cause of stragglers. ElasticNet addresses Lasso's limitations by grouping correlated variables and including the group when any of its variables predicts latency.

Bagging methods enable robust regression by avoiding over-fitted models, which do not accurately generalize beyond the training dataset. Bagging methods mitigate over-fitting by resampling the dataset [13]. The method constructs R replicas of a d-element dataset. Each replica draws d samples with replacement from the original dataset. The final model is a linear combination of R models fit for the replicas. Hound performs bagging on the dataset, uses ElasticNet to fit models to each replica, and reports the models' average.

Dependence Modeling (DP). Statistical dependence, a powerful framework for causal discovery [47], assesses the association between latency and profiled metrics. Table 1 presents properties

Dependence Measure	BP	NL	TI	Complexity
Pearson's ρ [22]	✓	✗	✗	$O(N)$
Spearman's ρ [22]	✗	✓	✓	$O(N\log(N))$
Kendall's τ [22]	✗	✓	✓	$O(N\log(N))$
Schweizer-Wolff Dependence (SWD) [5]	✓	✓	✓	$O(N^2)$

Table 1. Comparison of statistical dependence measures.

for various dependence measures [5]. First, measures should satisfy basic properties (BP) esablished by the first four of Rényi's classic axioms [53]. Moreover, they should be non-linear (NL) because dependences between performance and system conditions are often non-linear [63]. Third, measures should be invariant to strictly increasing transformations (TI), which are used to obfuscate industrial traces [50]. Our analysis favors the Schweizer-Wolff Dependence (SWD), but this measure is unsigned and computationally expensive. We address SWD's limitations and create the Signed Schweizer-Wolff Dependence (SSW).

SWD measures the dependence between two random variables X, Y. The joint distribution of these variables comprises two pieces of information— marginals and dependence. The Sklar Theorem separates them and defines Copula C to describe dependence [43]. SWD transforms variables using their cumulative distribution functions F_X, F_Y to obtain $u{\equiv}F_X(x)$, $v{\equiv}F_Y(y)$. The variables are increasingly dependent as the distance between $C(u,v)$ and $u{\cdot}v$ grows. SWD measures this distance as follows [47].

$$\text{SWD}(X,Y) = \frac{12}{N^2-1}\sum_{i=1}^{N}\sum_{j=1}^{N}|\widehat{C}(u_i,v_j)-u_i\cdot v_j|$$

$$C(u_i,v_j) = \frac{\#(x_k,y_k)\ s.t.\ x_k\le x_i,\ y_k\le y_j}{N},\ k{\in}[1,N]$$

Note that $(x_1,y_1),(x_2,y_2),\ldots$ are observed pairs of X, Y.

When dependence between two random variables is positive (negative), an increase in one suggests an increase (decrease) in the other. We use the Fréchet-Hoeffding Bound [43] to determine the sign. When a positive dependence grows stronger, $C(u,v)$ approaches $\min(u,v)$. When a negative dependence grows stronger, $C(u,v)$ approaches $\max(u+v-1,0)$. When X, Y are independent, $C(u,v)$ is equidistant between these bounds. SSW estimates the Copula's distance to its bounds to determine the sign within some ϵ.

$$\text{SSW}(X,Y) = \text{sign}(X,Y)\cdot\text{SWD}(X,Y)$$

$$\text{sign}(X,Y) = \begin{cases} -1 & \text{L}_1-\text{L}_2 > \epsilon \\ 0 & |\text{L}_1-\text{L}_2|\le\epsilon \\ 1 & \text{L}_2-\text{L}_1 < \epsilon \end{cases}$$

$$L_1 = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(\min(u_i, v_j) - \hat{C}(u_i, v_j)\right)$$

$$L_2 = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(\hat{C}(u_i, v_j) - \max(u_i + v_j - 1, 0)\right)$$

SSW values in [-1,0) indicate negative dependence, in (0,1] indicate positive dependence, and equal to 0 indicate independence. The SSW estimator has complexity $O(N^2)$ because the SWD and sign estimators visit all pairs (u_i, v_j) for $i, j \in [1, N]$. Sampling reduces cost.

Causal Modeling (CA). Confounding bias is a major challenge when seeking causal associations between latency and profiled metrics [7]. Confounding bias arises when the supposed causal association between two variables is partially or completely explained by a third, confounding variable. For example, suppose task latency is much higher on older processors than on newer ones. We cannot definitively say processor design causes the latency difference. Servers with new processors often use faster memory and the difference could be partially or primarily caused by memory design. Determining the causal effect of processors requires eliminating memory's effect.

Hound's CA learner constructs Rubin Causal Models (RCM) [55]. RCM estimates the causal effect of one metric on latency while eliminating bias induced by other metrics. Let Z be a binary random variable for a treatment level such as high and low processor usage. Let R be a continuous random variable for the response level such as high and low latency. We estimate the causal effect of Z on R while controlling for all other metrics X such as memory usage, scheduling events, etc. RCM measures the effect Δ, which is the difference in responses with and without treatment (*i.e.*, $R_1 - R_0$), for each task.

$$\Delta = \mathbb{E}(R_1 - R_0) = \mathbb{E}(R_1) - \mathbb{E}(R_0)$$

RCM uses Inverse Probability Weighting (IPW) to estimate Δ [36]. It estimates the effect despite missing data. For each data point, the treatment is either applied or not and the outcome is either R_1 or R_0. The data cannot report outcomes with and without treatment.

$$\Delta = \mathbb{E}\left[\frac{ZR}{e(X)}\right] - \mathbb{E}\left[\frac{(1-Z)R}{1-e(X)}\right]$$

$$e(X) = P\{Z = 1|X\}$$

Propensity score $e(X)$ is the conditional probability of having a treatment Z given values of confounding metrics X [55]. Given the score, outcomes and treatment are independent such that $\mathbb{E}(R_1) = \mathbb{E}[ZR/e(X)]$ and $\mathbb{E}(R_0) = \mathbb{E}[(1-Z)R/1-e(X)]$ [36]. This formulation resolves the missing value problem because Z and R are observed and $e(X)$ can be estimated from data.

We design AdaBoost Inverse Probability Weighting (AIPW) to estimate the propensity score. Propensity scores are usually estimated with logistic regression, $e(X) = \exp(\beta \cdot X)/(1 + \exp(\beta \cdot X))$. But this approach is vulnerable to collinearity, over-fitting, and outliers [48], which distort estimates of causal effects [36]. We address these challenges by enhancing IPW with AdaBoost [27], which estimates conditional probabilities more accurately than regression [65].

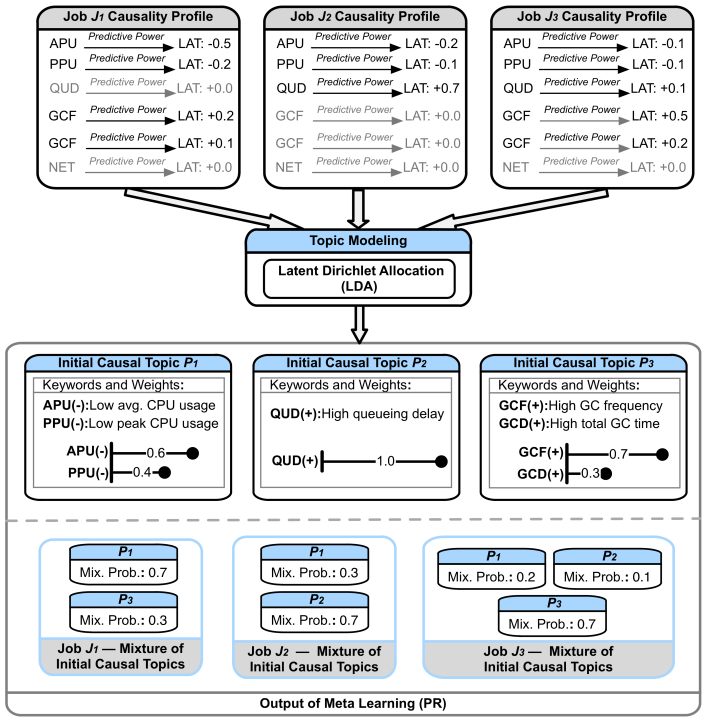


Fig. 5. Example - Hound Meta Learning

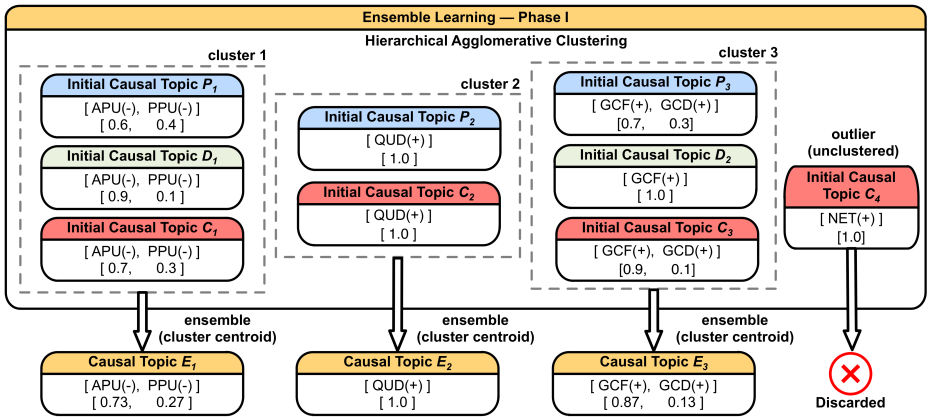
3.2 Meta Learning

Hound's meta learner uses topic models to identify patterns and extract semantic structure from jobs' numerous models and causality profiles. We use topic models because of the parallels between inferring themes in documents and identifying patterns in causality [9]. Topics arise from recurring clusters of words just as causes arise from recurring clusters of atypical metric values. Documents contain multiple topics just as profiles reveal causes of stragglers. Significant topics appear in many documents just as significant causes explain stragglers in many jobs.

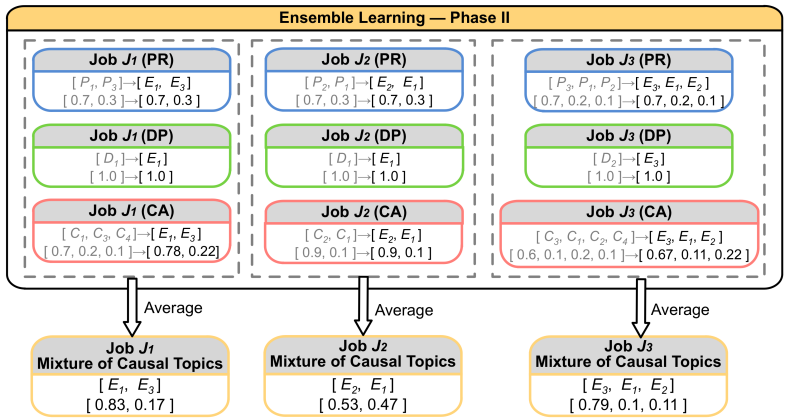
Hound uses Latent Dirichlet Allocation to infer causes of stragglers as shown in Figure 5. First, Hound populates a dictionary with metrics and identifies metrics that often appear together in jobs' causality profiles. Metrics that are more prominent in these profiles produce corresponding words that appear more frequently in documents. Topic P_1 is defined by APU(-) and PPU(-), which cluster within profiles. APU(-) is more significantly associated with latency and is weighted more heavily. Second, Hound identifies a mix of relevant topics for each job. Job J_2's stragglers are explained by topics P_1 and P_2. Topic P_2 is weighted more heavily because most stragglers are caused by queueing delay QUD(+) even though some are caused by low processor utilization.

3.3 Ensemble Learning

Hound's ensemble learner reconciles potentially divergent topics from multiple learners. The ensemble emphasizes topics found by multiple learners and drops those found by only one. Figure 6 illustrates this process. First, Hierarchical Agglomerative Clustering (HAC) identifies similarities in predictive (P_*), dependence (D_*), and causal (C_*) topics [42]. Although the ensemble identifies



(a) Clustering base topics to identify ensemble topics



(b) Assigning ensemble topics to individual jobs

Fig. 6. Example - Hound Ensemble Learning

consensus around some topics, such as low processor usage, APU(-) and PPU(-), it also corrects errors and outliers. P_* and C_* report queuing delay, QUD(+), as a cause when D_* misses it. C_* misreports network overhead, NET(+), as a cause when P_* and D_* do not. Clusters' centroids define the ensemble's topics. Second, Hound identifies relevant ensemble topics for each job. Weights for ensemble topics are averages of those for prediction, dependence, and causation topics.

4 EXPERIMENTAL METHODS

We implement Hound in Apache Spark with 3,800 lines of Python code. We deploy Hound on a cluster of eight NUMA nodes for parallel computing. Each node is configured with 48 AMD Opteron 6174 cores and 256GB DDR3 memory.

Task Metric	Description
Machine Capacity	
MACHINE_CPU	Num. cores in host machine
MACHINE_RAM	RAM in host machine (B)
Scheduling	
SCHED_DELAY	Task scheduling delay (μs)
PRIORITY	Task priority (integer)
EVICT	Num. times task is evicted
FAIL	Num. times task fails
Resource Request	
REQ_CPU	Num. processor cores requested
REQ_MEM	Amt. memory requested (B)
REQ_DISK	Amt. local storage requested (B)
Resource Usage	
PEAK_CPU	Max processor used (core-sec/sec)
MEAN_CPU	Mean processor used (core-sec/sec)
MEM_ASSIGN	Amt. memory allocated (B)
PEAK_MEM	Max memory used (B)
MEAN_MEM	Mean memory used (B)
PAGE_CACHE	Total page cache used (B)
PAGE_CACHE_UM	Unmapped page cache used (B)
PEAK_IO	Max I/O rate (I/O-sec/sec)
MEAN_IO	Mean I/O rate (I/O-sec/sec)
DISK_SPACE	Local storage used (B)
μarchitecture	
CPI	Cycles per instruction
CACHE_MISS	LLC misses per kilo-inst.

Table 2. Task metrics in the Google datacenter trace

4.1 Google Trace

We use a month-long trace from a production Google cluster that contains 12K servers and computes for diverse jobs such as web services, MapReduce, and high performance computing [49]. A job is a collection of tasks that execute the same program on different shards of data. Jobs are heterogeneous and configured with different scheduling levels. At one end of the spectrum are online (level-3) jobs that serve revenue generating user requests. At the other end are batch (level-0) jobs that support internal software development and background computation. The trace supplies 180GB of data for 650K jobs comprised of 25M tasks. Our analysis includes only production jobs, which comprise 12% of the total. We exclude jobs with limited task parallelism (*i.e.*, fewer than 20 tasks), which contribute 4.8% of the trace's tasks. After such post-processing, the trace contains 13K jobs and 3.3M tasks. Some tasks need treatment of missing values before statistical analysis.

Table 2 presents the metrics profiled for each task. At coarse-grain, system monitors track each task's scheduled processor time, allocated memory capacity, and actual resource utilization. At fine-grain, hardware counters track instruction throughput and cache miss rates. As tasks run, profilers report mean and max for resource usage and microarchitectural activity over a measurement period (*i.e.*, 300s). We average these periodic measurements over the task's duration. Collectively, these metrics can diagnose stragglers and reveal causes related to systems management such as resource allocation, job scheduling, colocation control, fail-over mechanisms.

4.2 Spark Traces

We use Spark traces to supplement the Google trace. Although we can validate our casual explanations for the Google trace, we cannot compare against expert diagnoses, which are not publicly available for this system. For this reason, we demonstrate Hound on a pair of traces collected for Spark performance on Amazon EC2 clusters and use a prior study of this data [46] to define expert diagnoses for comparison.

The first trace profiles a five-node Amazon EC2 cluster running Big Data Bench (BDBench), which launches five iterations of ten unique SQL queries [66]. The trace includes 84 Spark jobs, 162 stages, and 115K tasks that process 60GB of input data. The second trace profiles a twenty-node Amazon EC2 cluster running TPC-DS, which simulates 13 users executing 20 unique SQL queries in random order [45]. The trace includes 1,003 Spark jobs, 1,296 stages and 239K tasks that process 850GB of input data. We consider versions of BDBench and TPC-DS that compute on data stored on disk.

Table 3 presents the major performance metrics profiled for each task. Note that metrics are summary counters instead of time series data. Data is drawn from fine-grained instrumentation of Spark's run-time engine. Spark-SQL transforms SQL queries into Spark jobs [4]. A Spark job consists of multiple stages and each stage consists of one or more parallel tasks. Tasks within the same stage run the same binary with different partitions of the data. Typically, early stages read data from the file system or memory cache while later stages read the previous stage's data with a network shuffle [46].

5 EVALUATION WITH GOOGLE TRACE

Table 4 presents Hound's topics that explain stragglers in Google's datacenter. Topics constructed from prediction (P_*), dependence (D_*), and causation (C_*) models cluster to produce an ensemble topic (E_*). Each topic is a weighted combination of keywords, which are measured metrics followed by a suffix, drawn from models. The suffix "(+)" or "(-)" means the metric is significantly larger or smaller for stragglers than for normal tasks. Keywords and topics produce natural interpretations and reveal causes of stragglers. Many of the following causes are concise and consistent with domain-specific expertise, which is remarkable given the wealth of data.

Load Imbalance (Skew[2]). Hound finds that tasks with more work are more likely to be stragglers. Tasks that use more memory (E_0), larger page caches (E_1), or more local storage (E_2) may be computing on more data. These topics indicate larger working sets that may not fit in memory. Tasks that require more processor time (E_3) or I/O transactions (E_4) may also have more work.

Resource Constraints. Tasks that use fewer resources are likely to perform poorly. Underutilization of one resource suggests contention for others. Atypically low processor and memory use suggest poor progress (E_5, E_6), and atypically low I/O rates suggest constrained data supply (E_7, E_8). Note that atypically low and high processor usage are both causes for stragglers (E_3, E_5), but these conditions do not arise simultaneously. Some jobs' stragglers may be caused by low usage and other jobs' by high usage. Hound identifies causes from patterns across many jobs and then assigns the most relevant cause to each job.

Microarchitectural Activity. Processor counters shed light on poor task performance. Cache misses increase the average number of cycles required per instruction (E_9). Although low processor usage and frequent cache misses may appear correlated, Hound identifies separate causes (E_5 versus E_9) that are justified by domain expertise. When measuring processor usage, the operating system tracks processor time whereas the processor tracks cycles committing instructions. Processor

[2]Here, skew means the non-uniform partition of data across tasks rather than the statistical measure of asymmetry for a probability distribution.

Task Metric	Description
Executor	
EXE_DES_TIME	Executor deserialization time
EXE_RUN_TIME	Executor run time
BROADCAST_TIME	Variable broadcast time
Output	
RESULT_SIZE	Result size
RESULT_DES_TIME	Result deserialization time
OUTPUT_WRITE_TIME	Output blocked write time
OUTPUT_BYTES	Output size in bytes
Spill	
SPILL_BYTES	Number of bytes spilled to disk if data does not fit in memory
SPILL_BYTES_DISK	Number of bytes spilled to disk with compression
Input	
READ_BYTES	Total input size in bytes
READ_TIME	Total time to read input
HDFS_READ_BYTES	Size of input read from HDFS in bytes
HDFS_READ_PACKETS	Size of input read from HDFS in packets
HDFS_READ_TIME	Time to read input from HDFS
HDFS_OPEN_TIME	Time to open input file on HDFS
Shuffle Read	
SHFL_READ_RBYTES	Size of shuffle read from network in bytes
SHFL_READ_RBLKS	Size of shuffle read from network in blocks
SHFL_READ_WAIT_TIME	Time to open shuffle read connection
SHFL_READ_LBLKS	Size of shuffle read on local server in blocks
SHFL_READ_LBYTES	Size of shuffle read on local server in bytes
SHFL_READ_LTIME	Time to shuffle read on local server
Shuffle Write	
SHFL_WRITE_BYTES	Size of shuffle write in bytes
SHFL_WRITE_TIME	Time to shuffle write
SHFL_WRITE_OPEN_TIME	Time to open file for shuffle write
SHFL_WRITE_CLOSE_TIME	Time to close file for shuffle write
Processor Usage	
CPU_USER	User-space CPU usage
CPU_SYS	Kernel-space CPU usage
Disk Usage	
DISK_UTILIZATION	Disk utilization
DISK_READ_THPT	Disk read throughput
DISK_WRITE_THPT	Disk write throughput
Network Usage	
NET_READ_BYTES_PSEC	Number of network bytes read per second
NET_SEND_BYTES_PSEC	Number of network bytes sent per second
NET_READ_PACK_PSEC	Number of network packets read per second
NET_SEND_PACK_PSEC	Number of network packets sent per second
Scheduler Delay	
SCHED_DELAY	Time taken by scheduler to place a task to worker
Garbage Collection	
GC_TIME	Garbage collection overhead
First Task	
FIRST_TASK	True if no task of the same stage finished on the same worker

Table 3. Task metrics in the Spark traces for BDBench and TPC-DS

time can be low, perhaps due to thread scheduling, even when pipelines rarely stall for caches. Conversely, processor time can be high even when tasks have poor instruction-level parallelism.

Topic	Keywords	Weights	Cluster	Interpretation
E_0	MEM_ASSIGN(+), MEAN_MEM(+), PEAK_MEM(+)	0.5, 0.25, 0.25	P_0, P_3, D_0, C_0	Data Skew
E_1	PAGE_CACHE(+), PAGE_CACHE_UM(+), MEM_ASSIGN(+)	0.45, 0.38, 0.17	P_1, D_1, C_1	Data Skew
E_2	DISK_SPACE(+)	1.0	P_2, D_2, C_2	Data Skew
E_3	MEAN_CPU(+), PEAK_CPU(+)	0.52, 0.48	P_4, D_3, C_3	Computation Skew
E_4	PEAK_IO(+), MEAN_IO(+)	0.51, 0.49	P_5, D_4, C_4	I/O Skew
E_5	MEAN_CPU(-), PEAK_CPU(-)	0.8, 0.2	P_6, D_5, C_5, C_6	Limited Processor
E_6	MEAN_MEM(-), PEAK_MEM(-)	0.83, 0.17	P_7, D_6, D_7, C_7	Limited Memory
E_7	MEAN_IO(-)	1.0	D_8, C_8	Limited I/O
E_8	PEAK_IO(-), MEAN_IO(-)	0.83, 0.17	P_8, D_9	Limited I/O
E_9	CACHE_MISS(+), CPI(+)	0.54, 0.46	P_9, D_{10}, C_9	Cache Bottleneck
E_{10}	SCHED_DELAY(+)	1.0	P_{10}, D_{11}, C_{10}	Scheduler (Queueing) Delay
E_{11}	EVICT(+)	1.0	P_{11}, C_{11}	Eviction Delay
P_{12}	FAIL(+)	1.0	unclustered	✖
C_{12}	MACHINE_RAM(+)	1.0	unclustered	✖

Table 4. Hound's causal topics for the Google dataset, derived from an ensemble of predictive (P), dependent (D), causal (C) models. Cluster identifies models' topics that produce each ensemble topic. Appendix B shows individual models' topics. Topics with "✖" are revealed by individual models but rejected by the ensemble.

Job ID	Causes	Weight
6283499093	Data Skew	1.00
6266469130	Queueing Delay	1.00
6274140245	Limited Processor	1.00
6343946350	Limited Processor Limited Memory	0.65 0.35
6308689702	Data Skew Computation Skew I/O Skew	0.53 0.27 0.20
6343048076	Data Skew Eviction Queueing Delay Limited Processor Limited I/O	0.33 0.25 0.18 0.12 0.12

Table 5. Example - Mixtures of causes

Scheduling Problems. Finally, Hound finds that the cluster manager's decisions affect task completion. Queueing delays cause stragglers by extending tasks' end-to-end latency (E_{10}). Eviction delays also cause stragglers as tasks halt computation on overloaded machines, re-launch on another machine, and lose progress (E_{11}).

Hound's ensemble guards against false conclusions, which may be consistent with operator intuition but arise from biased models. Systems operators might think that machine heterogeneity creates stragglers as some tasks run on slower machines, but only the causation model flags memory heterogeneity (C_{12}). Operators might worry about transient task failures that require re-launch, but only the prediction model raises this concern (P_{12}). Users appropriately size processor and memory requests such that they are irrelevant in our analysis of stragglers.

Number of Causes	1	2	3	4	5
Percent of Jobs	12%	33%	36%	16%	3%

Table 6. Number of causes per job

Cause	**Coverage**	**Dominant Coverage**
Data Skew (E_0, E_1, E_2)	73.6%	55.0%
Limited Processor (E_5)	39.2%	12.1%
Cache Misses (E_9)	32.6%	7.0%
Limited I/O (E_7, E_8)	36.7%	6.6%
Queueing Delay (E_{10})	20.0%	5.1%
Limited Memory (E_6)	13.6%	2.7%
Computation Skew (E_3)	31.2%	2.2%
Eviction Delay (E_{11})	3.80%	0.90%
I/O Skew (E_4)	5.60%	0.60%

Table 7. Coverage statistics for causes

5.1 Mixtures of Causes

Hound not only identifies recurring causes of stragglers, it determines which jobs suffer from which causes. Hound assigns each job a weighted mix of causes. Weights estimate the fraction of stragglers attributed to each cause in the mix as shown in Table 5 for representative jobs. Sometimes, a job's stragglers can be explained by a single cause; job 682. . .093's stragglers are explained by data skew. More often, a job's stragglers have multiple causes; job 634. . .350's stragglers are explained by a mix of low processor and memory utilization. Hound guards against faulty mixtures of causes by constructing an ensemble of independent models (*i.e.*, for prediction, dependence, and causation). Appendix C shows how ensemble corrects faulty causal explanations produced by a single model.

We propose a series of measures to understand mixtures of causes. We say a cause is *dominant* when it explains the majority of a job's stragglers. We define a cause's *coverage* as how often it explains some of a job's stragglers. We define a cause's *dominant coverage* as how often it explains the majority of a job's stragglers, which is a more selective measure and smaller than coverage. Topics with greater coverage are relevant for more jobs and provide greater utility to system operators.

Consider Table 5 for examples. Job 630. . .702's dominant cause is data skew since its weight is greater than the sum of weights for computation and I/O skew. As seen for job 634. . .076, not every job has a dominant cause. Suppose the system has only the six jobs in Table 5, Data skew's coverage is 50% because it explains three of six jobs while limited processor and memory's coverage are 33.3% and 16.7%, respectively. Data skew's dominant coverage is 33.3% whereas limited memory's is zero.

Table 6 presents the percentage of jobs that require multiple causes. Although Hound identifies eleven possible causes, each used by some jobs, more than 80% of jobs require fewer than three causes to explain their stragglers. When multiple causes are required, one is usually dominant.

Table 7 summarizes coverage. The top three causes—data skew, limited processor utilization, cache misses—explain the majority of stragglers in 74.1% of jobs. 73.6% of jobs experience some form of data skew. Many jobs also suffer from low resource usage or cache behavior. Relatively minor causes include memory availability, task eviction and I/O skew.

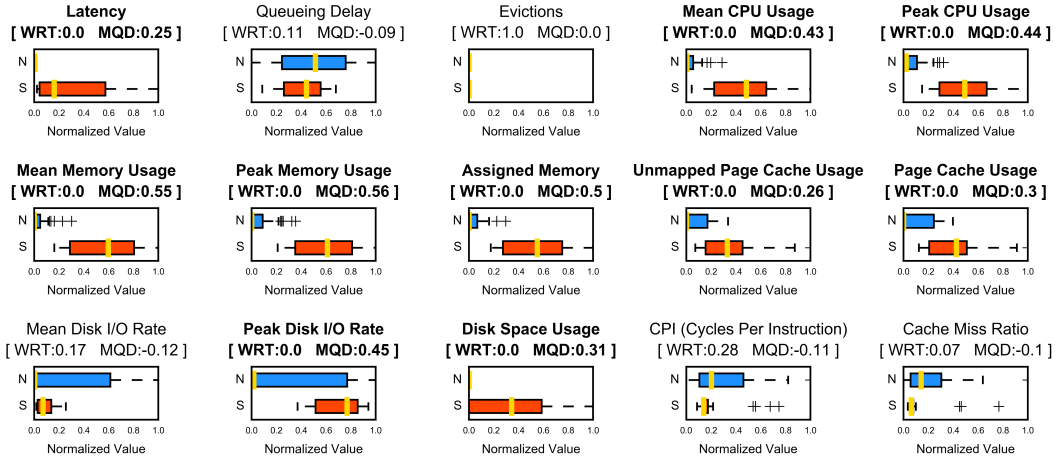


Fig. 7. Profiles suggest stragglers in job 6308689702 arise from a mix of data, computation, and I/O skew. Hound's inferred causes match human analysis. "S" denotes the distribution of values for stragglers, which is plotted in red; "N" denotes the distribution of values, which is plotted in blue; yellow lines indicate medians in the distributions.

5.2 Validation with Case Studies

We determine whether Hound's causal explanations are consistent with domain expertise by comparing inferred causes with manually diagnosed ones. First, we use boxplots to visualize distributions of microarchitectural, system, and scheduling activity. Visual differences in boxplots for stragglers (S) and normal tasks (N) suggest causes of stragglers.

Second, we measure differences between stragglers and normal tasks' data distributions by calculating the Wilcoxon Ranksum Test (WRT [67]) and the Mean Quartile Difference (MQD). WRT evaluates a null hypothesis in which profiled distributions for stragglers and normal tasks exhibit no significant difference. Rejected hypotheses, when WRT < 0.05, reveal causes of stragglers. MQD averages differences between two distributions' quartiles. Suppose datasets d_S, d_N are profiled from stragglers and normal tasks, and suppose Q_i denotes a dataset's i-th quantile. MQD calculates

$$\frac{1}{3}\sum_{i=\{25,50,75\}} Q_i\left(d_S\right) - Q_i\left(d_N\right)$$

We perform manual diagnosis for a series of case studies. First, we classify tasks within a job as straggling or normal based on measured latency. Then, we determine whether stragglers' metrics differ from normal tasks' because those that differ may explain stragglers' atypical latency. Finally, we compare this manual analysis against Hound's inferred causes. Figure 7 considers a job with stragglers due to multiple causes. Stragglers use more memory, page cache, and disk space. They also require more processor time and have higher peak I/O rates. Manual analysis is challenging because data indicate multiple atypical behaviors.

Inferred causes are consistent with domain expertise. Hound automatically infers three causes—data, computation, and I/O skew. Moreover, it accurately infers the relative importance of these causes. Hound assigns weights of 0.53, 0.27, and 0.20 to data, computation, and I/O skew. These weights are consistent with manual analysis. MQD for data skew's metrics are larger than computation and I/O skew's. Data skew explains more stragglers than other types of skew.

Straggler Cause	Example
Data Skew	Hash function based partitioning scheme distributes data unevenly across tasks. Tasks processing more data are likely to be stragglers. [32]
Resource Constraints	Processor [2], memory [71] and I/O [23] are contended by collocated tasks, background daemons, garbage collector, etc.
μarchitecture Activity	Colocated tasks contend for shared hardware such as cache. [74]
Scheduler Delay	Tasks trapped in multiple layers of queues in servers and network switches. [23]
Eviction Delay	Low-priority tasks are preempted, re-launch on another machine, and lose progress. [74]
Computation Skew	Some records (e.g., dense graphs) are much more computationally expensive than other records (e.g., sparse graphs). [31]
I/O Skew	Stragglers are caused due to intensive disk I/O, such as Spark's shuffle write. [46]
Network Congestion	A rack with many tasks can be congested on its network link and produce stragglers. [2]
Hardware Heterogeneity	Tasks assigned to obsolete hardware can be slower than others. [72]
Power Management	Power-saving can add significant delay when moving from inactive to active modes. [23]

Table 8. Examples of stragglers' causes from related studies that produce expert diagnoses.

Study	Data Skew	Resource Constraints	μarch. (Cache)	Scheduler Delay	Eviction Delay	Computation Skew	I/O Skew	Network* Congestion	H/W* Hetero.	Power* Manage.
G-1[23]	–	✓	✓	✓	–	–	–	✓	–	✓
G-2[74]	–	–	✓	–	✓	–	–	–	–	–
DS-1[32]	✓	–	–	–	–	✓		–	–	–
DS-2[2]	✓	✓	–	–	–	–	–	✓	–	–
DS-3[12]	✓	✓	–	–	–	–	–	–	–	–
DS-4[31]	✓	–	–	–	–	✓	–	–	–	–
DS-5[72]	–	–	–	–	–	–	–	–	✓	–
DS-6[21]	✓	✓	–	–	–	–	–	–	–	–
DS-7[46]	✓	✓		✓	–	–	✓	–	–	–
DS-8[71]	✓	✓	–	–	–	–	–	–	–	–
Hound	✓	✓	✓	✓	✓	✓	✓	–	–	–

Table 9. Comparison of causes diagnosed by Hound for the Google system against causes previously diagnosed by experts for related systems. "G-" denotes a prior study of Google datacenters; "DS-" denotes a prior study on distributed systems such as Hadoop and Spark."*" identifies system causes that are beyond this paper's scope because the Google trace lacks relevant profiles.

Our findings are borne out repeatedly for the datacenter's diverse jobs. Appendix D presents case studies for additional jobs. Appendix E assesses causal diagnosis from an information-theoretic analysis on all jobs. We argue that Hound's inferred causes are accurate because task latency has high mutual information with metrics included in causal topics and low mutual information with all other metrics. And we argue that its causes are coherent because metrics within a causal topic have high mutual information with each other and low mutual information otherwise. Thus, Hound automates model inference and data analysis yet identifies causes that are accurate, coherent, and consistent with manual analysis.

5.3 Comparisons with Expert Diagnosis

Tables 8-9 survey studies in which experts discover and investigate specific causes of stragglers in large-scale, distributed systems. Hound discovers causes in the Google system that match causes found by expert diagnoses in several other systems. It discovers the same fundamental system phenomena that were discovered in other studies despite differences in methods and systems.

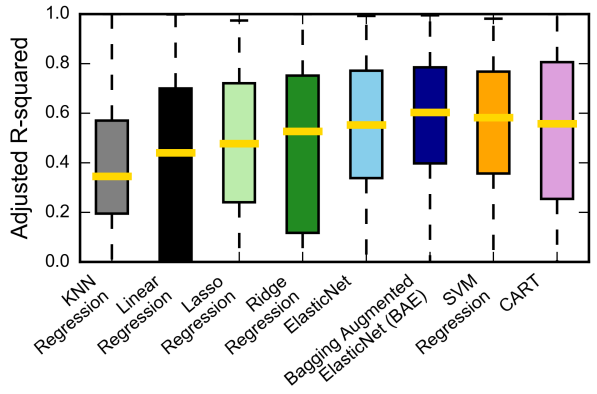


Fig. 8. Comparison of modeling methods for latency prediction on the Google dataset.

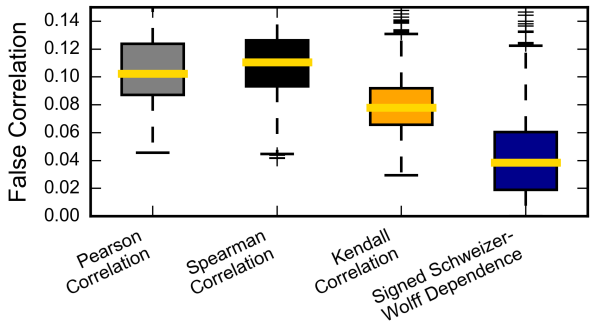


Fig. 9. Comparison of dependence measures for safeguards against false correlation on the Google dataset.

The only gaps in diagnosis relate to network congestion, hardware heterogeneity and power management; the Google trace lacks profiles to reveal these conditions. Moreover, Hound identifies prevalent causes of stragglers. Prior studies often attribute stragglers to data skew and resource constraints, which match Hound's diagnoses. Coverage statistics indicate that data skew and bottlenecks in processors, memory, and I/O are dominant causes for 76% of the datacenter's jobs (*cf.* Table 7).

One might be concerned about comparing Hound for a Google datacenter and assorted methods for other systems, but the comparison is informative. Google's trace includes both interactive and batch jobs. Hound's analysis of interactive jobs matches other Google-based studies of tails in latency-sensitive services [23, 74], and our analysis of batch jobs is consistent with those for Hadoop and Spark. Second, Google's trace, with over 40K unique logical job names (*i.e.*, binaries), is diverse and representative of real-world workloads [49].

5.4 Comparison with Simpler Base Learners

Predictive Models. Hound constructs a model to predict task latency from system conditions. Figure 8 compares Hound's method against state-of-the-art predictive models by illustrating the distribution of adjusted R^2 values across all jobs. Hound's Bagging Augmented ElasticNet

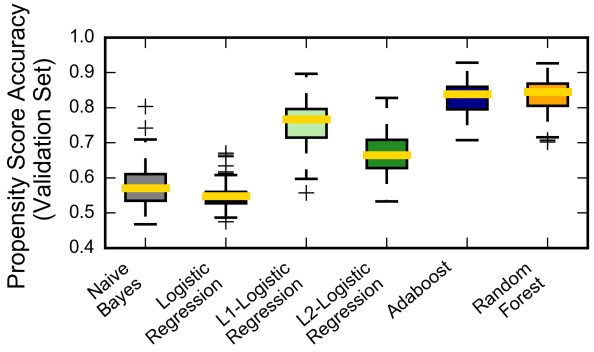


Fig. 10. Comparison of propensity score estimates for causal inference on the Google dataset.

(BAE) performs 16% better than linear regression, slightly better than ElasticNet and CART, and comparably to SVM. Nearest neighbor regression (KNN) and linear regression perform much worse than others. Linear regression shows negative coefficients for more than 25% of jobs. Appendix F shows how linear regression produces faulty causal explanations.

Dependence Models. Copula-based dependence measures control false correlations better than conventional measures such as Pearson's correlation [35, 47]. Figure 9 shows dependence estimates after we destroy the correlation between system conditions and task latency by randomly shuffling latency measurements in the dataset. Better estimators are those that more consistently report a value close to zero since there is no dependence between variables. Hound's Signed Schweizer-Wolff (SSW) guards against false correlations much better than Pearson, Spearman, and Kendall's measures. Appendix F shows how Pearson's estimates produce faulty causal explanations.

Causal Models. The core of the Rubin Causal Model (RCM) is the propensity score and we compare state-of-the-art methods for estimating these scores (*cf.* Figure 10). Hound's AdaBoosted IPW estimates are most accurate. Naive Bayes and logistic regression perform much worse than others. Lasso regularization performs in between. Appendix F shows how logistic regression for propensity score estimation produces faulty causal explanations.

6 EVALUATION WITH SPARK TRACES

Hound reproduces the results of domain-specific expertise. Ousterhout et al. say something causes stragglers if tasks become normal when that thing requires no time [46]. Such expert diagnosis for BDBench produces four observations. First, three causes—HDFS Read (Disk), Garbage Collection, Shuffle Write (Disk)—explain a large fraction of stragglers. Second, one cause—Output Size (Skew)—explains a small fraction of stragglers. Third, other events—First Task, Scheduler Delay, Shuffle Read—are insignificant causes. Finally, no cause among the three most prominent ones dominate. Table 10 presents Hound's eight topics, which map to four straggler causes that are consistent with expert diagnosis. Table 11 indicates that mixture probabilities, which estimate the number of stragglers explained by each cause, also align with expert diagnosis.

Similarly, expert diagnosis for TPC-DS produces five observations. First, two causes—HDFS Read(Disk), Output Size(Skews)—explain a large fraction of stragglers. Second, three causes—Shuffle Read(Network), Garbage Collection, Shuffle Write(Disk)—explain a moderate fraction of stragglers. Third, one cause—Scheduler Delay—explains a small fraction of stragglers. First Task is an insignificant cause. Finally, there is no dominant cause among the two most prominent causes and no dominant cause among the next three most prominent causes. Tables 12-13 present Hound's

Topic Keywords	Weights	Interpretation	Mean Mixture Probability
DISK_READ_THPT(+)	1.0	HDFS (Input) read	16%
READ_BYTES(+), READ_TIME(+), HDFS_READ_BYTES(+), HDFS_READ_TIME(+)	0.25,0.25, 0.25,0.25	HDFS (Input) read	10%
HDFS_READ_TIME(+), READ_TIME(+), HDFS_OPEN_TIME(+)	0.35,0.35,0.3	HDFS (Input) read	10%
GC_TIME(+)	1.0	GC overhead	30%
DISK_UTILIZATION(+), DISK_WRITE_THPT(+)	0.5, 0.5	Shuffle write	14%
SHFL_WRITE_OPEN_TIME(+)	1.0	Shuffle write	9%
SHFL_WRITE_TIME(+), SHFL_WRITE_CLOSE_TIME(+)	0.62,0.38	Shuffle write	2%
OUTPUT_WRITE_TIME(+), DISK_WRITE_TIME(+), OUTPUT_BYTES(+)	0.35,0.33, 0.32	Output skew	9%

Table 10. Hound's causal topics for the Spark BDBench dataset

HDFS (Input) Read	GC Overhead	Shuffle Write	Output Skew
36%	30%	25%	9%

Table 11. Hound's estimate of stragglers (percentage) explained by each cause for the Spark BDBench dataset

Topic Keywords	Weights	Interpretation	Mean Mixture Probability
HDFS_READ_TIME(+), INPUT_READ_TIME(+)	0.5, 0.5	HDFS (Input) read	23%
HDFS_READ_BYTES(+)	1.0	HDFS (Input) read	8%
RESULT_SIZE(+)	1.0	Output skew	23%
SHFL_READ_WAIT_TIME(+)	1.0	Shuffle read	17%
DISK_UTILIZATION(+)	1.0	Shuffle write	8%
SHFL_WRITE_OPEN_(+), SHFL_WRITE_TIME(+), SHFL_WRITE_CLOSE_TIME(+)	0.34,0.33, 0.33	Shuffle write	4%
DISK_WRITE_THPT(+)	1.0	Shuffle write	4%
GC_TIME(+)	1.0	GC Overhead	10%
SCHED_DELAY(+)	1.0	Scheduler delay	3%

Table 12. Hound's causal topics for the Spark TPC-DS dataset

HDFS (Input) Read	Output Skew	Shuffle Read	Shuffle Write	GC overhead	Scheduler Delay
31%	23%	17%	16%	10%	3%

Table 13. Hound's estimate of stragglers (percentage) explained by each cause for the Spark TPC-DS dataset

analysis. Nine topics map to six straggler causes that are consistent with expert diagnosis. Mixture probabilities are also broadly consistent with expert diagnosis.

7 COMPLEXITY AND OVERHEADS

Hound's approximate complexity is $O(MN)$, where M and N represent the number of tasks per job and number of jobs per trace. Although the number of tasks per job can be as large as a several thousand, very few jobs have such a high degree of parallelism. In practice, the number of tasks is order of magnitudes larger than the number of jobs.

Figure 11 presents measured scalability, indicating that Hound's run time increases linearly with trace size. Each measurement is the average of five runs on randomly drawn Google jobs. Moreover,

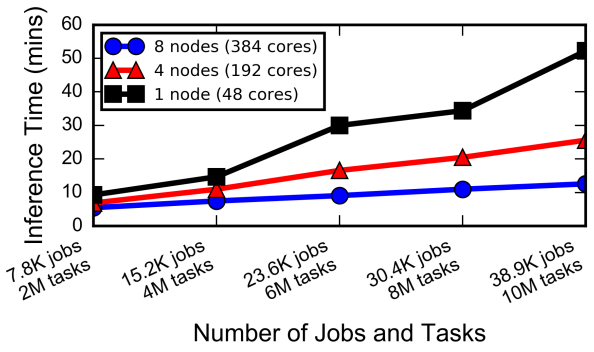


Fig. 11. Hound scalability as dataset size increases.

we measure scalability with respect to the number of workers used for learning. We implement Hound with Apache Spark to parallelize learning on massive datasets. Given 38.9K jobs and 10M tasks, Hound's needs 52 minutes to complete inference on a single worker and just 12 minutes on eight workers.

8 RELATED WORK

8.1 Straggler Mitigation

Speculative Execution. Dean et al. detect when a task runs slower than expected and launches another equivalent task as a backup, hoping that transient performance issues disappear [24]. But a few limitations of speculative execution have been identified in prior research. First, speculative execution is too aggressive in heterogeneous clusters as too many speculative tasks are launched and reduce system capacity. LATE [72] addresses this problem by throttling the number of backup tasks and by replicating only a few prioritized slow tasks rather than any slow one. Second, as a reactive approach, speculative execution is usually too late to help small, latency-sensitive jobs. Dolly [1] improves timeliness by proactively launching multiple clones of every task of a job and using only the clone finishes first. Third, speculative execution improves the performance of the job at hand but can hurt the performance of others since backup tasks can cause resource contention. Hopper [52] addresses contention by right sizing the resource pool for backup tasks.

Scheduling. Schedulers can mitigate stragglers by preventing tasks from computing on slow machines. Mantri [2] improves task placement to avoid congested network link. Wrangler [70] predicts the straggler risk for each machines and risky machines are prohibited from serving certain tasks.

Root Cause Analysis. *Tales of Tail* [34] experimentally explores whether stragglers are caused by interference from background daemons, poor scheduling, constrained concurrency models, or power throttling. Treadmill [75] carries out experiments to measure tail latencies under different hardware configurations (NUMA, Turbo Boost, DVFS and NIC) and use Analysis of Variance (ANOVA) to estimate the impact of different hardware factors. Determining and validating the suspected causes from *Tales of Tail* and Treadmill require deep expertise in the specific system and architecture. In comparison, Hound is a statistical machine learning framework that focuses on automated causal discovery rather than experimental validation. In addition, Hound is independent of any specific system or architecture detail.

8.2 Performance Analysis

Machine Learning. Learning has been widely applied in prior research to help understand performance issues in large-scale distributed system. Related methods can be classified into *regression-*, *tree-*, and *graph*-based methods. Regression-based methods (*e.g.*, Highlighter [10] and Fingerprint[11]) use regression models, such as Lasso, to correlate the state of service-level-obligations (SLO) with system metrics. The constructed model can select a few salient metrics, usually called the signature, that help operators understand anomalies such as SLO violations.

Similarly, tree-based methods [56, 69] apply classification and regression trees to characterize the causal rules that produce specific performance outcomes. Graph-based methods [19, 41, 61] apply probabilistic graphs, such as Bayesian networks, to visualize the causal relationship between system metrics and performance state. In comparison, Hound performs causal learning with little domain knowledge and constructs topic models that emphasize interpretability, reliability and scalability.

Tracing Techniques. Profilers and tracers for large-scale distributed systems include Pivot Tracing [37], *The Mystery Machine* [17], Magpie [8], *lprof* [76], Pinpoint [16], XTrace [26] and Dapper [58]. These frameworks stitch dispersed event logs from many thousands of machines and reconstruct the complete control flow (system path) for each user request. In comparison, Hound focuses on trace analysis not acquisition.

9 CONCLUSION AND FUTURE WORK

Hound is a statistical machine learning framework for diagnosing stragglers at datacenter scale. Hound offers interpretability, reliability and scalability. We apply Hound to analyze a production Google datacenter and two experimental Amazon EC2 clusters, revealing challenges and providing insights for future systems design and management.

For future work, Hound is an open framework and could incorporate additional causal analysis algorithms, such as Recursive Structural Equation Models (RSEM) [14], as base learners for even more reliable inference. Moreover, Hound assumes a unified profiling framework that reports the same metrics for every job and task. When profilers are heterogeneous, Hound will require new methods for inferring causal topics from different vocabularies.

ACKNOWLEDGMENTS

The authors would like to thank our referees and shepherd for their valuable comments and helpful suggestions. This work is supported by the National Science Foundation under grants CCF-1149252 (CAREER), CCF-1337215 (XPS-CLCCA), SHF-1527610, and AF-1408784. This work is also supported by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of these sponsors.

REFERENCES

[1] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. 2013. Effective Straggler Mitigation: Attack of the Clones. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI '13)*. 185–198. http://dl.acm.org/citation.cfm?id=2482626.2482645

[2] Ganesh Ananthanarayanan, Srikanth Kandula, Albert G Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. 2010. Reining in the Outliers in Map-Reduce Clusters using Mantri.. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI '10)*, Vol. 10. 24.

[3] J. Anderson, L. Berc, J. Dean, S. Ghemawat, M. Henzinger, S. Leung, R. Sites, M. Vandervoorde, C. Waldspurger, and W. Weihl. 1997. Continuous Profiling: Where have all the cycles gone?. In *Proc. Symposium on Operating Systems Principles (SOSP)*.

[4] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al. 2015. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 1383–1394.

[5] E. F. Wolff B. Schweizer. 1981. On Nonparametric Measures of Dependence for Random Variables. *The Annals of Statistics* 9, 4 (1981), 879–885. http://www.jstor.org/stable/2240856

[6] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. 2013. Developing a predictive model of quality of experience for internet video. In *ACM SIGCOMM Computer Communication Review (SIGCOMM'13)*, Vol. 43. ACM, 339–350.

[7] Elias Bareinboim and Judea Pearl. 2011. Controlling Selection Bias in Causal Inference. In *AAAI*.

[8] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. 2004. Using Magpie for Request Extraction and Workload Modelling.. In *6th USENIX Symposium on Operating Systems Design and Implementation*, Vol. 4. 18–18.

[9] David M Blei. 2012. Probabilistic topic models. *Commun. ACM* 55, 4 (2012), 77–84.

[10] Peter Bodík, Moises Goldszmidt, and Armando Fox. 2008. HiLighter: Automatically Building Robust Signatures of Performance Behavior for Small- and Large-scale Systems. In *Proceedings of the Third Conference on Tackling Computer Systems Problems with Machine Learning Techniques*. USENIX Association, Berkeley, CA, USA, 3–3.

[11] Peter Bodik, Moises Goldszmidt, Armando Fox, Dawn B. Woodard, and Hans Andersen. 2010. Fingerprinting the Datacenter: Automated Classification of Performance Crises. In *EuroSys 2010*. 111–124.

[12] Edward Bortnikov, Ari Frank, Eshcar Hillel, and Sriram Rao. 2012. Predicting execution bottlenecks in map-reduce clusters. In *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*. USENIX Association, 18–18.

[13] Leo Breiman. 1996. Bagging predictors. *Machine learning* 24, 2 (1996), 123–140.

[14] Carlos Brito and Judea Pearl. 2012. Graphical condition for identification in recursive SEM. *arXiv preprint: 1206.6821* (2012).

[15] Philip K. Chan and Salvatore J. Stolfo. 1993. Experiments on Multistrategy Learning by Meta-learning. In *CIKM '93*. 314–323. https://doi.org/10.1145/170088.170160

[16] Mike Y Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric Brewer. 2002. Pinpoint: Problem determination in large, dynamic internet services. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*. IEEE, 595–604.

[17] Michael Chow, David Meisner, Jason Flinn, Daniel Peek, and Thomas F Wenisch. 2014. The mystery machine: End-to-end performance analysis of large-scale internet services. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*. 217–231.

[18] Ira Cohen, Moises Goldszmidt, Terence Kelly, Julie Symons, and Jeffrey S. Chase. 2004. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. In *6th USENIX Symposium on Operating Systems Design and Implementation (OSDI '04)*. 16–16. http://dl.acm.org/citation.cfm?id=1251254.1251270

[19] Ira Cohen, Steve Zhang, Moises Goldszmidt, Julie Symons, Terence Kelly, and Armando Fox. 2005. Capturing, Indexing, Clustering, and Retrieving System History. In *20th ACM Symposium on Operating Systems Principles (SOSP '05)*. 105–118. https://doi.org/10.1145/1095810.1095821

[20] Gregory F. Cooper. 1990. The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks (Research Note). *Artif. Intell.* 42, 2-3 (March 1990), 393–405. https://doi.org/10.1016/0004-3702(90)90060-D

[21] Henggang Cui, James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Abhimanu Kumar, Jinliang Wei, Wei Dai, Gregory R. Ganger, Phillip B. Gibbons, Garth A. Gibson, and Eric P. Xing. 2014. Exploiting Bounded Staleness to Speed Up Big Data Analytics. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference (USENIX ATC'14)*. USENIX Association, Berkeley, CA, USA, 37–48. http://dl.acm.org/citation.cfm?id=2643634.2643639

[22] Suzana de Siqueira Santos, Daniel Yasumasa Takahashi, Asuka Nakata, and André Fujita. 2013. A comparative study of statistical methods used to identify dependencies between gene expression signals. *Briefings in bioinformatics* (2013), 051.

[23] Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. *Commun. ACM* 56, 2 (2013), 74–80.

[24] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *CACM* 51, 1 (Jan. 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[25] Carsten F Dormann, Jane Elith, Sven Bacher, Carsten Buchmann, Gudrun Carl, Gabriel Carré, Jaime R García Marquéz, Bernd Gruber, Bruno Lafourcade, Pedro J Leitão, et al. 2013. Collinearity: a review of methods to deal with it and a simulation study evaluating their performance. *Ecography* 36, 1 (2013), 27–46.

[26] Rodrigo Fonseca, George Porter, Randy H Katz, Scott Shenker, and Ion Stoica. 2007. X-trace: A pervasive network tracing framework. In *Proceedings of the 4th USENIX conference on Networked systems design & implementation*. USENIX Association, 20–32.

[27] Yoav Freund and Robert E Schapire. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. System Sci.* 55, 1 (Aug. 1997), 119–139. https://doi.org/10.1006/jcss.1997.1504

[28] S. Ghemawat, H. Gobioff, and S. Leung. 2003. The Google File System. In *Proc. Symposium on Operating Systems Principles (SOSP '03)*.
[29] S. Graham, P. Kessler, and M. McKusick. 1982. Gprof: A call graph execution profiler. In *Proc. Symposium on Compiler Construction (CC)*.
[30] Matthew Hoffman, Francis R. Bach, and David M. Blei. 2010. Online Learning for Latent Dirichlet Allocation. In *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (Eds.). Curran Associates, Inc., 856–864.
[31] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. 2010. Skew-resistant parallel processing of feature-extracting scientific user-defined functions. In *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 75–86.
[32] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. 2011. A Study of Skew in MapReduce Applications. In *The 5th International Open Cirrus Summit*.
[33] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. 2012. SkewTune: Mitigating Skew in Mapreduce Applications. In *SIGMOD '12*. 25–36. https://doi.org/10.1145/2213836.2213840
[34] Jialin Li, Naveen Kr. Sharma, Dan R. K. Ports, and Steven D. Gribble. 2014. Tales of the Tail: Hardware, OS, and Application-level Sources of Tail Latency. In *SOCC '14*. Article 9, 14 pages. https://doi.org/10.1145/2670979.2670988
[35] David Lopez-Paz, Philipp Hennig, and Bernhard Schölkopf. 2013. The randomized dependence coefficient. In *Advances in neural information processing systems*. 1–9.
[36] Jared K Lunceford and Marie Davidian. 2004. Stratification and weighting via the propensity score in estimation of causal treatment effects: a comparative study. *Statistics in medicine* 23, 19 (2004), 2937–2960.
[37] Jonathan Mace, Ryan Roelke, and Rodrigo Fonseca. 2015. Pivot tracing: dynamic causal monitoring for distributed systems. In *25th ACM Symposium on Operating Systems Principles (SOSP '09)*. 378–393.
[38] Ajay Anil Mahimkar, Zihui Ge, Aman Shaikh, Jia Wang, Jennifer Yates, Yin Zhang, and Qi Zhao. 2009. Towards Automated Performance Diagnosis in a Large IPTV Network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication (SIGCOMM '09)*. ACM, New York, NY, USA, 231–242. https://doi.org/10.1145/1592568.1592596
[39] Carl Mela and Praveen Kopalle. 2002. The impact of collinearity on regression analysis: the asymmetric effect of negative and positive correlations. *Applied Economics* 34, 6 (2002), 667–677.
[40] Jesús Muñoz and Ángel M Felicísimo. 2004. Comparison of statistical methods commonly used in predictive modelling. *Journal of Vegetation Science* 15, 2 (2004), 285–292.
[41] Karthik Nagaraj, Charles Killian, and Jennifer Neville. 2012. Structured comparative analysis of systems logs to diagnose performance problems. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI '12)*. 353–366.
[42] Mirco Nanni. 2005. Speeding-up hierarchical agglomerative clustering in presence of expensive metrics. In *PAKDD '05*. Springer, 378–387.
[43] Roger B Nelsen. 2007. *An Introduction to Copulas*. Springer Science & Business Media.
[44] Sebastian Ordyniak and Stefan Szeider. 2010. Algorithms and Complexity Results for Exact Bayesian Structure Learning. In *UAI 2010*.
[45] Raghunath Othayoth and Meikel Poess. 2006. The making of tpc-ds. In *PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES*, Vol. 32. 1049.
[46] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. 2015. Making Sense of Performance in Data Analytics Frameworks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, 293–307.
[47] Barnabás Póczos, Zoubin Ghahramani, and Jeff G Schneider. 2012. Copula-based Kernel Dependency Measures. In *ICML '12*. 775–782.
[48] Daryl Pregibon. 1982. Resistant fits for some commonly used logistic models with medical applications. *Biometrics* (1982), 485–498.
[49] Charles Reiss and John Wilkes. 2011. Google cluster-usage traces: format+ schema. *Technical Report* (2011).
[50] Charles Reiss, John Wilkes, and Joseph L Hellerstein. 2012. Obfuscatory obscanturism: making workload traces of commercially-sensitive systems safe to release. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 1279–1286.
[51] Gang Ren, Eric Tune, Tipp Moseley, Yixin Shi, Silvius Rus, and Robert Hundt. 2010. Google-wide profiling: A continuous profiling infrastructure for data centers. *IEEE Micro* 4 (2010), 65–79.
[52] Xiaoqi Ren, Ganesh Ananthanarayanan, Adam Wierman, and Minlan Yu. 2015. Hopper: Decentralized Speculation-aware Cluster Scheduling at Scale. In *SIGCOMM '15*. 379–392. https://doi.org/10.1145/2785956.2787481
[53] Alfréd Rényi. 1959. On measures of dependence. *Acta mathematica hungarica* 10, 3-4 (1959), 441–451.
[54] Irina Rish, Mark Brodie, Sheng Ma, Natalia Odintsova, Alina Beygelzimer, Genady Grabarnik, and Karina Hernandez. 2005. Adaptive diagnosis in distributed systems. *IEEE Transactions on neural networks* 16, 5 (2005), 1088–1109.

[55] Paul R Rosenbaum and Donald B Rubin. 1983. The central role of the propensity score in observational studies for causal effects. *Biometrika* 70, 1 (1983), 41–55.
[56] Raja R. Sambasivan, Alice X. Zheng, Michael De Rosa, Elie Krevat, Spencer Whitman, Michael Stroucken, William Wang, Lianghong Xu, and Gregory R. Ganger. 2011. Diagnosing Performance Changes by Comparing Request Flows. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI'11)*. USENIX Association, Berkeley, CA, USA, 43–56.
[57] C. Shannon and W. Weaver. 1949. *The mathematical theory of communication.* University of Illinois Press.
[58] Benjamin H Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. 2010. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure. (2010).
[59] Harald Steck. 2008. Learning the Bayesian Network Structure: Dirichlet Prior versus Data. In *UAI 2008*.
[60] Jiang Su and Harry Zhang. 2006. A fast decision tree learning algorithm. In *UAI*. AAAI Press, 500–505.
[61] Mukarram Tariq, Amgad Zeitoun, Vytautas Valancius, Nick Feamster, and Mostafa Ammar. 2008. Answering what-if deployment and configuration questions with wise. In *ACM SIGCOMM Computer Communication Review*, Vol. 38. ACM, 99–110.
[62] Eno Thereska, Bjoern Doebel, Alice X Zheng, and Peter Nobel. 2010. Practical performance models for complex, popular applications. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 38. ACM, 1–12.
[63] Eno Thereska and Gregory R Ganger. 2008. IRONModel: Robust performance models in the wild. *ACM SIGMETRICS Performance Evaluation Review* 36, 1 (2008), 253–264.
[64] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society.* (1996), 267–288.
[65] Jelte Peter Vink and Gerard de Haan. 2015. Comparison of machine learning techniques for target detection. *Artificial Intelligence Review* 43, 1 (2015), 125–139.
[66] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, et al. 2014. Bigdatabench: A big data benchmark suite from internet services. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on.* IEEE, 488–499.
[67] Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics bulletin* 1, 6 (1945), 80–83.
[68] D. H. Wolpert and W. G. Macready. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1, 1 (Apr 1997), 67–82. https://doi.org/10.1109/4235.585893
[69] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the 22nd Symposium on Operating Systems Principles (SOSP'09)*. ACM, 117–132.
[70] Neeraja J. Yadwadkar, Ganesh Ananthanarayanan, and Randy Katz. 2014. Wrangler: Predictable and Faster Jobs Using Fewer Resources. In *SOCC '14*. Article 26, 14 pages. https://doi.org/10.1145/2670979.2671005
[71] Neeraja J. Yadwadkar, Bharath Hariharan, Joseph E. Gonzalez, and Randy Katz. 2016. Multi-Task Learning for Straggler Avoiding Predictive Job Scheduling. *Journal of Machine Learning Research* 17, 106 (2016), 1–37.
[72] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. 2008. Improving MapReduce Performance in Heterogeneous Environments. In *8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*. 29–42. http://dl.acm.org/citation.cfm?id=1855741.1855744
[73] Steve Zhang, Ira Cohen, Julie Symons, and Armando Fox. 2005. Ensembles of Models for Automated Diagnosis of System Performance Problems. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN '05)*. IEEE Computer Society, Washington, DC, USA, 644–653. https://doi.org/10.1109/DSN.2005.44
[74] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. 2013. CPI2: CPU Performancfe Isolation for Shared Compute Clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys '13)*. ACM, New York, NY, USA, 379–391. https://doi.org/10.1145/2465351.2465388
[75] Yunqi Zhang, David Meisner, Jason Mars, and Lingjia Tang. 2016. Treadmill: Attributing the Source of Tail Latency through Precise Load Testing and Statistical Inference. In *ISCA '16*.
[76] Xu Zhao, Yongle Zhang, David Lion, Muhammad Faizan Ullah, Yu Luo, Ding Yuan, and Michael Stumm. 2014. lprof: A non-intrusive request flow profiler for distributed systems. In *11th USENIX Symposium on Operating Systems Design and Implementation*. 629–644.
[77] Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society* 67, 2 (2005), 301–320.

A LEARNING ALGORITHMS

Algorithm 1–4 describe the learning algorithms in Section 3.

ALGORITHM 1: Bagging Augmented ElasticNet (BAE)

- **Input:**
  1. Dataset $\mathcal{D} = (\mathbf{X_n}, \mathbf{y_n})$. Matrix $\mathbf{X_n}$ and vector $\mathbf{y_n}$ denote the observed task metrics and latencies, respectively, for all tasks in job J_n.
  2. Number of bootstrap replicates, I.
- **Initialize:** Create I bootstrap replicates $D_1, D_2, \ldots, D_I$ from $\mathcal{D}$.
- **For** $i = 1, 2, \ldots, I$:
  Train an ElasticNet model on dataset D_i and apply LARS-EN algorithm to estimate the coefficients $\hat{\beta}^{(i)}$:

$$\underset{\hat{\beta}^{(i)}}{\text{argmin}} \; L(\hat{\beta}^{(i)}) = \|\mathbf{y_n} - \mathbf{X_n}\hat{\beta}^{(i)}\|_2^2 + \lambda_1\|\hat{\beta}^{(i)}\|_2^2 + \lambda_2\|\hat{\beta}^{(i)}\|_1$$

- **Output:** the average of coefficient estimates $\hat{\beta} = \frac{1}{I}(\hat{\beta}^{(1)} + \hat{\beta}^{(2)} + \ldots \hat{\beta}^{(I)})$

ALGORITHM 2: AdaBoost Propensity Score Estimation

- **Input:**
  1. Observations $(x_1, z_1), (x_2, z_2), \ldots, (x_M, z_M)$ for (X, Z), $X \in [0,1]^{P\text{-}1}$, $Z \in \{-1, 1\}$. x_m indicates the values of confounding metrics (X) observed from task m, and z_m indicates the value of the treatment metric (Y) observed from task m.
  2. A weak binary classification algorithm $H : X \to Z$, which predicts the treatment metric (Z) with the confounding metrics (X).
  3. Initial sampling distribution $D = [D_0, D_1, \ldots, D_M]$, with D_m indicating the sampling probability for observation (x_m, z_m).
  4. The number of iterations I.
- **Initialize:** $D(1) = D$
- **For** $i = 1, 2, \ldots, I$
  1. Train a weak classifier h_i using H with samples drawn using $D(i)$.
  2. Estimate the error rate of training, $\epsilon_i = (\#\ of\ x_m\ s.t.\ h_i(x_m) \neq z_m)/M$.
  3. Determine the weight for weak classifier h_i: $\alpha_i = \frac{1}{2}log(\frac{1-\epsilon_i}{\epsilon_i})$.
  4. Create a new sampling distribution $D(i+1)$ from $D(i)$ with α_i:
$$D_m(i+1) = D_m(i) \cdot e^{-\alpha_i \cdot z_m \cdot h_i(x_m)} \quad (m = 1, 2, \cdots, M)$$
  5. Normalize the new sampling distribution:
$$D_m(i+1) = \frac{D_m(i+1)}{S_{i+1}} \quad (m = 1, 2, \cdots, M), \quad S_{i+1} = \sum_{m=1}^{M} D_m(i+1)$$
- **Boosted Hypothesis** $f(X) = \sum_{i=1}^{I} \alpha_i h_i(X)$.
- **Output:** $\widehat{e}_{AdaBoost}(X) = P\{Z = 1|X\} = 1/(1 + e^{-2f(X)})$.

ALGORITHM 3: Meta Learning Algorithm

- **Input:**
  1. Causality profiles $\mathcal{C}$=$\{C_1, C_2, \ldots, C_N\}$
  2. Minimum number of topics k_{min} (default k_{min}=5)
  3. Stopping criterion ϵ (minimum document coverage for a topic, default ϵ=0.05)
  4. Trimming threshold η (default η=0.1)
- **Initialize:**
  1. **Create vocabulary** $\mathcal{V}$. Each task metric K_i ($i = 1, \ldots, P$) corresponds to two *anomalous words*, K_i (+) and K_i (-), which indicate a straggler's value for K_i is higher and lower than normal, respectively. $\mathcal{V}$ is a vocabulary containing all anomalous words, $\mathcal{V} = \{K_1(+), K_1(-), \ldots, K_P(+), K_P(-)\}$.
  2. **Create documents** $\mathcal{D}$. Each document $D_i \in \mathcal{D}$ ($i = 1, \ldots, N$) is created with causality profile C_i.
     (a). For each entry $C_{i \cdot j}$ in C_i ($j = 1, 2, \ldots, P$), calculate probabilities $Pr\{K_j(+)\}$, $Pr\{K_j(-)\}$:
     $$\begin{cases} Pr\{K_j(+)\} = |C_{i \cdot j}|,\ Pr\{K_j(-)\} = 0 & C_{i \cdot j} > 0 \\ Pr\{K_j(+)\} = 0,\ Pr\{K_j(-)\} = |C_{i \cdot j}| & C_{i \cdot j} < 0 \\ Pr\{K_j(+)\} = 0,\ Pr\{K_j(-)\} = 0 & C_{i \cdot j} = 0 \end{cases}$$
     (b). $Pr\{K_1(+)\}, Pr\{K_1(-)\}, Pr\{K_2(+)\}, Pr\{K_2(-)\}, \ldots, Pr\{K_P(+)\}, Pr\{K_P(-)\}$ form a discrete distribution R_i. Randomly draw W (default W=500) words from R_i to create document D_i.
- **For** $k = k_{min},\ k_{min} + 1,\ k_{min} + 2, \ldots$
  1. **Infer** topics β=$\beta_1, \beta_2, \ldots, \beta_k$, and the probabilistic mixture of topics for each job θ=$\{\theta_1, \theta_2, \ldots, \theta_N\}$ from D with Latent Dirichlet Allocation (LDA).
  2. **Trim** each topic to eliminate trivial words (with weight lower than η), and trim the topic mixture for each document to remove trivial topics (with probability lower than η). After trimming, rescale β_i and every $\theta_i \in \theta$ to keep each of them sum-to-one.
  3. **Calculate** the document coverage for each topic. Suppose $\theta_{i \cdot j}$ is the mixture probability of the j-th topic in D_i. The document coverage of the i-th topic is $\left(\sum_{j=1}^{N} I\{\theta_{i \cdot j} > 0\}\right) / N$, where I is indicator function.
  4. **Stop** if any inferred topic has document coverage lower than *epsilon*.
- **Output** current β, θ. Each topic in β is defined as an ***Initial Causal Topic***.

ALGORITHM 4: Ensemble Learning Algorithm

Phase I

- **Input:**
  1. Initial causal topics $\mathcal{P}$={$P_1, \ldots, P_{Q_1}$}, $\mathcal{D}$={$D_1, \ldots, D_{Q_2}$}, $\mathcal{C}$={$C_1, \ldots, C_{Q_3}$}, by PR, DP, CA, respectively.
  2. Mixture of initial causal topics for each job, $\mathcal{PP}$={$PP_1, PP_2, \ldots, PP_N$}, $\mathcal{PD}$={$PD_1, PD_2, \ldots, PD_N$}, $\mathcal{PC}$={$PC_1, PC_2, \ldots, PC_N$}, by PR, DP, CA, respectively.
  3. Minimum and maximum number of causes, m_1, m_2.
  4. Trimming threshold η (default η=0.1).
- **Initialize:** $SCORE \leftarrow \mathbf{0}$
- **For** $m = m_1, m_1{+}1, m_1{+}2, \ldots, m_2$
  1. Apply HAC (Hierarchical Agglomerative Clustering) to cluster all of the initial causal topics $\mathcal{P} \cup \mathcal{D} \cup \mathcal{C}$. Set the target number of clusters as m.
  2. Calculate score $SCORE[m]$ for the m clusters. Let H_1 denote the average intra-cluster similarity, and H_2 denote the average inter-cluster similarity (default: *cosine* similarity, λ=2/3):

$$SCORE[m] \leftarrow \lambda H_1 + (1 - \lambda)(1 - H_2)$$

- Determine the optimal number of clusters $L = \text{argmax}_m\, SCORE[m]$. Let $\{S_1, S_2, \ldots, S_L\}$ be the clusters produced by HAC with optimal parameter L. Each cluster S_i ($i = 1, 2, \ldots, L$) is a set of one or more initial causal topics.
- For each cluster S_i ($i = 1, 2, \ldots, L$), take its centroid E_i as the ensemble of initial causal topics in S_i. If a cluster contains only a single topic or all its topics are produced by the same model (either PR, DP or CA), its centroid is defined as $\emptyset$.
- Define a set $E = \{E_i | E_i \neq \emptyset, i = 1, 2, \ldots, L\}$. Each element in E is considered as a ***Causal Topic***.

Phase II

- **Update the mixture of initial causal topics** in $\mathcal{PP}$ for each job to that of causal topics and similarly, update $\mathcal{PD}$ and $\mathcal{PC}$. Suppose $\mathcal{PP}$, $\mathcal{PD}$, and $\mathcal{PC}$ are updated to $\{PPE_n\}$, $\{PDE_n\}$, $\{PCE_n\}$ ($n = 1, 2, \ldots, N$), respectively.
- **Determine the ensemble of topic mixture** for each job. Define $PE_n = 1/3 \cdot (PPE_n + PDE_n + PCE_n)$ as the ensemble of topic mixture for job J_n. That is, PE_n is the mix of causal topics for J_n. Define $\mathcal{PE} = \{PE_1, PE_2, \ldots, PE_N\}$.
- **Trim mixture of causal topics** for each job to eliminate trivial topics (with weight lower than η). After trimming, rescale each $PE_i \in \mathcal{PE}$ to keep it sum-to-one.
- **Output** E and $\mathcal{PE}$

B CAUSAL TOPICS

Table 14, 15, 16 show causal topics inferred from the Google trace by predictive (PR), dependence (DP) and causal (CA) models, respectively.

Topic	Keywords	Weights	Interpretation
P_0	MEAN_MEM(+), PEAK_MEM(+), MEM_ASSIGN(+)	0.36, 0.34, 0.3	Data Skew
P_1	PAGE_CACHE(+), PAGE_CACHE_UNMAP(+)	0.55, 0.45	Data Skew
P_2	DISK_SPACE(+)	1.0	Data Skew
P_3	MEM_ASSIGN(+)	1.0	Data Skew
P_4	PEAK_CPU(+), MEAN_CPU(+)	0.55, 0.45	Computation Skew
P_5	MEAN_IO(+), PEAK_IO(+)	0.57, 0.43	I/O Skew
P_6	MEAN_CPU(-)	1.0	Limited Processor
P_7	MEAN_MEM(-)	1.0	Limited Memory
P_8	PEAK_IO(-), MEAN_IO(-)	0.66, 0.34	Limited I/O
P_9	CACHE_MISS(+), CPI(+)	0.55, 0.45	Cache Bottleneck
P_{10}	SCHED_DELAY(+)	1.0	Queueing Delay
P_{11}	EVICT(+)	1.0	Eviction Delay
P_{12}	FAIL(+)	1.0	Failure Delay

Table 14. Hound's causal topics for the Google dataset, discovered by predictive model (PR)

Topic	Keywords	Weights	Interpretation
D_0	MEAN_MEM(+), PEAK_MEM(+), MEM_ASSIGN(+)	0.36, 0.35, 0.29	Data Skew
D_1	PAGE_CACHE(+), PAGE_CACHE_UNMAP(+), MEM_ASSIGN(+)	0.44, 0.35, 0.21	Data Skew
D_2	DISK_SPACE(+)	1.0	Data Skew
D_3	MEAN_CPU(+), PEAK_CPU(+)	0.57, 0.43	Computation Skew
D_4	PEAK_IO(+), MEAN_IO(+)	0.55, 0.45	I/O Skew
D_5	MEAN_CPU(-), PEAK_CPU(-)	0.74, 0.26	Limited CPU
D_6	MEAN_MEM(-)	1.0	Limited Memory
D_7	PEAK_MEM(-), MEM_ASSIGN(-), MEAN_MEM(-), PAGE_CACHE_UNMAP(-), PAGE_CACHE(-)	0.3, 0.25, 0.2, 0.13, 0.12	Limited Memory
D_8	MEAN_IO(-)	1.0	Limited I/O
D_9	PEAK_IO(-)	1.0	Limited I/O
D_{10}	CACHE_MISS(+), CPI(+)	0.54, 0.46	Cache Bottleneck
D_{11}	SCHED_DELAY(+)	1.0	Queueing Delay

Table 15. Hound's causal topics for the Google dataset, discovered by dependence model (DP)

Topic	Keywords	Weights	Interpretation
C_0	PEAK_MEM(+), MEM_ASSIGN(+), MEAN_MEM(+) PAGE_CACHE(+), PAGE_CACHE_UNMAP(+)	0.26, 0.23, 0.23, 0.16, 0.12	Data Skew
C_1	PAGE_CACHE(+), PAGE_CACHE_UNMAP(+), MEM_ASSIGN(+)	0.36, 0.33, 0.31	Data Skew
C_2	DISK_SPACE(+)	1.0	Data Skew
C_3	MEAN_CPU(+), PEAK_CPU(+)	0.55, 0.45	Computation Skew
C_4	PEAK_IO(+), MEAN_IO(+)	0.56, 0.44	I/O Skew
C_5	MEAN_CPU(-)	1.0	Limited Processor
C_6	PEAK_CPU(-), MEAN_CPU(-)	0.53, 0.47	Limited Processor
C_7	MEAN_MEM(-)	1.0	Limited Memory
C_8	MEAN_IO(-)	1.0	Limited I/O
C_9	CACHE_MISS(+), CPI(+)	0.52, 0.48	Cache Bottleneck
C_{10}	SCHED_DELAY(+)	1.0	Queueing Delay
C_{11}	EVICT(+)	1.0	Eviction Delay
C_{12}	MACHINE_RAM (+)	1.0	Machine Heterogeneity

Table 16. Hound's causal topics for the Google dataset, discovered by causal model (CA).

Model	Causes	Weights
DP	Data Skew (D_0)	0.27
	Computation Skew (D_3)	0.26
	I/O Skew (D_4)	0.25
	Data Skew (D_1)	0.22
CA	Computation Skew (C_3)	0.36
	Data Skew (C_0)	0.33
	I/O Skew (C_4)	0.31
PR	Data Skew (P_0)	0.31
	Data Skew (P_2)	0.22
	Data Skew (P_1)	0.19
	Computation Skew (P_4)	0.17
	Limited I/O (P_8)	0.11
ENS	Data Skew ($E_0 + E_3 + E_2$)	0.51
	Computation Skew (E_9)	0.26
	I/O Skew (E_4)	0.19
	Limited I/O (E_6) (✖)	0.04

Table 17. Comparison of inferred causes from varied modeling strategies for job 6308689702.

Model	Causes	Weights
DP	Limited Processor (D_5)	0.35
	Limited Memory (D_6)	0.29
	Limited Memory (D_7)	0.19
	Cache Bottleneck (D_{10})	0.17
CA	Limited Processor (C_5)	1.0
PR	Limited Memory (P_7)	0.55
	Limited Processor (P_6)	0.45
ENS	Limited Memory (E_2)	0.6
	Limited Processor (E_0)	0.34
	Cache Bottleneck (E_5)(✖)	0.06

Table 18. Comparison of inferred causes from varied modeling strategies for job 6343946350.

C ENSEMBLE OF MODELS

Hound integrates topics inferred by an ensemble of independent models for predictive (PR), causal (CA), and dependence (DP) relationships. The ensemble reports causes revealed by multiple models and discards the rest. In Table 4, the ensemble eliminates task eviction, reported only by PR, and task failure, reported only by CA. Were it to use one model, Hound could discover misleading topics.

Tables 17–18 show how ensembles avoid faulty diagnoses. For job 630. . .702, DP and CA reveal a mix of data, computation, and I/O skew whereas PR alone identifies limited I/O resources. The ensemble considers I/O scarcity a false cause. For job 634. . .350's causes, DP alone reveals cache misses while CA misses limited memory. The ensemble considers cache behavior a false cause, but includes memory scarcity as a true cause.

D CASE STUDIES

Figure 12 compares profiles for stragglers and normal tasks. By definition, stragglers' latencies are several times larger than normal tasks'. Stragglers have significantly higher memory usage,

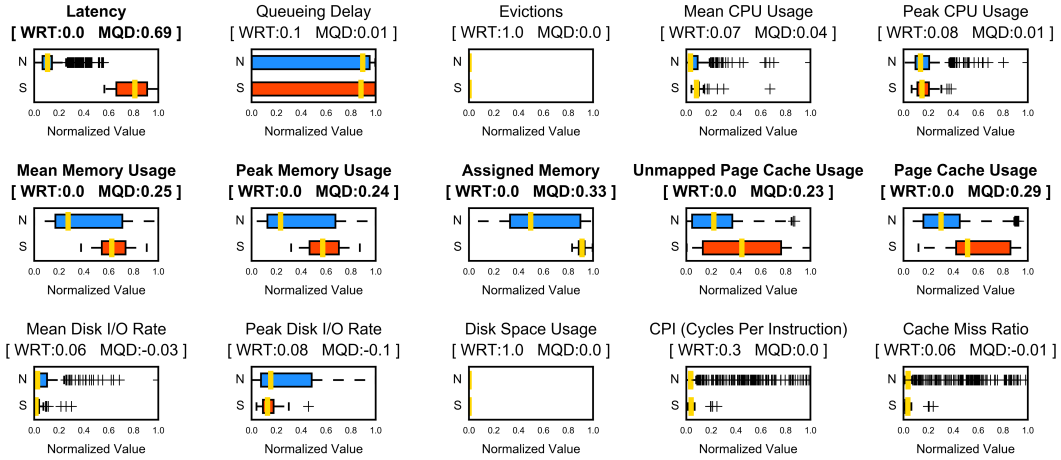


Fig. 12. Profiles suggest stragglers in job 6283499093 arise from data skew. Hound's inferred causes match human analysis. "S" denotes the distribution of values for stragglers, which is plotted in red; "N" denotes the distribution of values, which is plotted in blue; yellow lines indicate medians in the distributions.

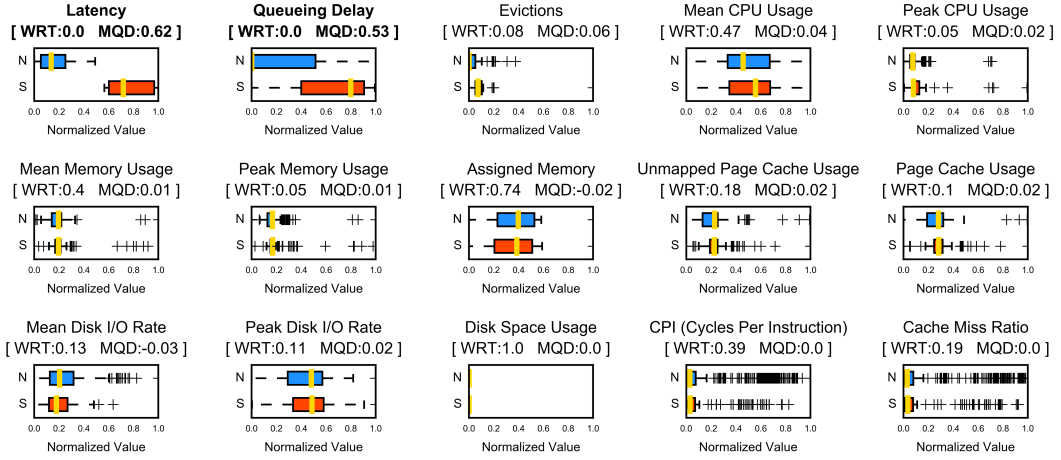


Fig. 13. Profiles suggest stragglers in job 6266469130 arise from queueing delay. Hound's inferred causes match human analysis. "S" denotes the distribution of values for stragglers, which is plotted in red; "N" denotes the distribution of values, which is plotted in blue; yellow lines indicate medians in the distributions.

memory allocations, and page cache usage. Manual analysis suggests data skew is a probable cause of stragglers, matching Hound's automatically inferred cause.

In Figure 13, stragglers have significantly higher queueing time than normal tasks. Other metrics show comparable distributions between stragglers and normal tasks. Manual analysis suggests queueing delay is a probable cause of stragglers, matching Hound's automatically inferred cause.

In Figure 14, stragglers have significantly lower processor usage than normal tasks. Other metrics show comparable distributions between stragglers and normal tasks. Manual analysis suggests limited processor resources is a probable cause of stragglers, matching Hound's automatically inferred cause.

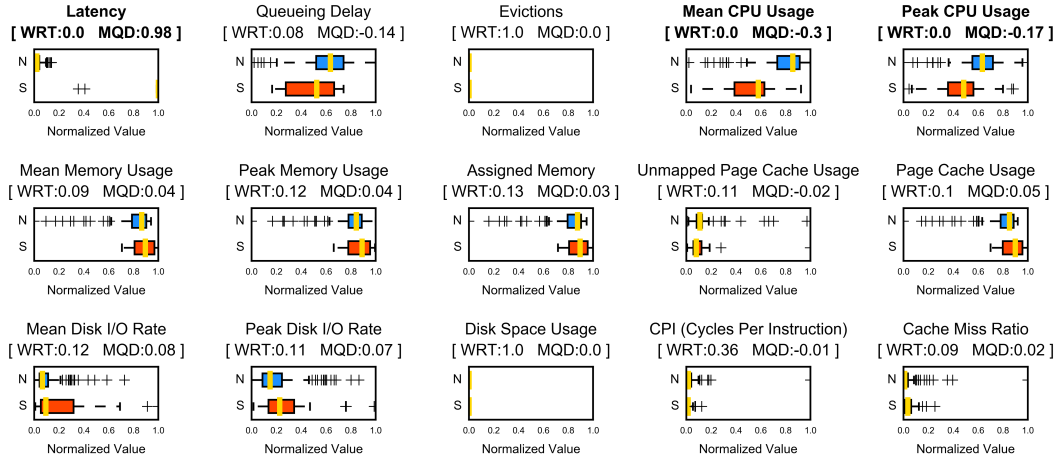


Fig. 14. Profiles suggest stragglers in job 6274140245 arise from limited processor usage. Hound's inferred causes match human analysis. "S" denotes the distribution of values for stragglers, which is plotted in red; "N" denotes the distribution of values, which is plotted in blue; yellow lines indicate medians in the distributions.

E MUTUAL INFORMATION

We draw on information theory to test the validity of Hound's topics. Shannon mutual information, in units of nats, quantifies information obtained about one random variable through another random variable [57]. The mutual information between discrete variables X and Y, with joint density $P_{XY}(x, y)$ and marginals $P_X(x)$ and $P_Y(Y)$, is

$$I(X;Y) = \sum_{x,y} P_{XY}(x,y) \log \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)}$$

.

We use mutual information to validate the accuracy and coherence of Hound's causal topics assigned for each job. For accuracy, task latency should have high mutual information with metrics within causal topics ($I_{M \cdot L}$) and low mutual information otherwise ($I_{U \cdot L}$). For coherence, metrics within a causal topic should have high mutual information with each other ($I_{M \cdot M}$) and low mutual information otherwise ($I_{M \cdot U}$). For each of these desiderata, we calculate mutual information, averaged over topics and metrics.

$$\begin{aligned}
I^{(j)}_{M \cdot L} &= \mathbb{E}_t \, \mathbb{E}_w \, I(M^{(j)}_{t,w};\ L^{(j)}) \\
I^{(j)}_{U \cdot L} &= \mathbb{E}_k \, I(U^{(j)}_{k};\ L^{(j)}) \\
I^{(j)}_{M \cdot M} &= \mathbb{E}_t \, \mathbb{E}_{v,w} \, I(M^{(j)}_{t,v};\ M^{(j)}_{t,w}) \\
I^{(j)}_{M \cdot U} &= x \, \mathbb{E}_t \, \mathbb{E}_{w,k} \, I(M^{(j)}_{t,w};\ U^{(j)}_{k})
\end{aligned}$$

where j identifies job, L denotes task latency, $M_{t,w}$ denotes the metric profiled for word w in a topic t assigned to the job, and U_k denotes a metric not in any topic.

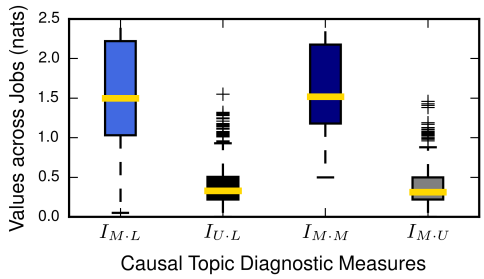


Fig. 15. Mutual information to assess accuracy (large I_{ML}, small I_{UL}) and coherence (large I_{MM}, small $_{UM}$). Boxplots show distribution of values across all jobs in the datacenter trace.

Topic	Keywords	Weights
E_0	PEAK_MEM(+), MEAN_MEM(-)	0.52, 0.48
E_1	MEAN_MEM(+), PEAK_MEM(-)	0.71, 0.29
E_2	MEAN_MEM(-), MEM_ASSIGN(+)	0.54, 0.46
E_3	CACHE_MISS(-), CPI(+)	0.5, 0.5
E_4	CACHE_MISS(+), CPI(-)	0.52, 0.48
E_5	PAGE_CACHE(+), PAGE_CACHE_UM(-)	0.66, 0.34
E_6	PAGE_CACHE(-), PAGE_CACHE_UM(+)	0.5, 0.5
E_7	MEM_ASSIGN(+)	1.0
E_8	DISK_SPACE(+)	1.0
E_9	MEAN_CPU(-)	1.0
E_{10}	SCHED_DELAY(+)	1.0
E_{11}	EVICT_DELAY(+)	1.0

Table 19. Causal topics inferred with linear regression as base learner.

Figure 15 shows the distribution of mutual information measures across all jobs in the datacenter trace. Large I_{ML} and I_{UL} indicate accuracy as Hound's topics include the most relevant metrics and exclude the rest. Large I_{MM} and small I_{UM} indicate coherence as Hound's topics integrate multiple metrics into a causal explanation for stragglers.

F SIMPLER BASE LEARNERS

Predictive Models. Table 19 indicates that multicollinearity affects topic models. Linear regression's topics, such as E_3 and E_4, report correlated metrics with contradictory signs. Model parameters with incorrect signs and implausible magnitudes are a typical symptom of multicollinearity [39]. BAE regularization mitigates these effects and produces better causal explanations.

Dependence Models. Table 20 indicates that Pearson's linear estimate affects topic models. First, Pearson causes models to miss important keywords in inferred topics, which often have just one keyword. Pearson's topics also include erroneous relationships, such as the negative correlation between scheduling delay and task latency in topic D_0; the correlation should in fact be positive. Pearson's correlation is sensitive to noise and outliers, which can create false correlations. Moreover, Pearson is a linear estimator and may miss non-linear associations.

Causal Models. Table 21 indicates that Rubin Causal Models that use logistic regression to estimate propensity scores are ineffective when inferring topics. Logistic regression severely distorts

Topic	Keywords	Weights
D_0	MEAN_IO(-),SCHED_DELAY(-), MEM_ASSIGN(-), RAM_REQ(-)	0.45, 0.19, 0.18, 0.18
D_1	MEM_ASSIGN(+)	1.0
D_2	SCHED_DELAY(+)	1.0
D_3	CACHE_MISS(+)	1.0
D_4	MEAN_CPU(-)	1.0
D_5	MEAN_MEM(-)	1.0
D_6	PAGE_CACHE(+), MEAN_MEM(+)	0.65, 0.35

Table 20. Causal topics inferred with Pearson's correlation as base learner.

Topic	Keywords	Weights
C_0	CPI(+), , CACHE_MISS(+)	0.5, 0.5
C_1	CPI(-)	1.0
C_2	PEAK_IO(+), MEAN_IO(+)	0.63, 0.37
C_3	MEAN_IO(+), PEAK_IO(-)	0.69, 0.31
C_4	MEAN_IO(-), PEAK_IO(-)	0.52, 0.48
C_5	PAGE_CACHE(-), PAGE_CACHE_UM(-)	0.53, 0.47
C_6	PAGE_CACHE(+), PAGE_CACHE_UM(-)	0.51, 0.49
C_7	PEAK_MEM(+), MEAN_MEM(+)	0.51, 0.49
C_8	MEAN_MEM(-), PEAK_MEM(+)	0.52, 0.48
C_9	RAM_CAPACITY(+)	1.0
C_{10}	RAM_CAPACITY(-)	1.0
C_{11}	SCHED_DELAY(+)	1.0
C_{12}	SCHED_DELAY(-)	1.0
C_{13}	MEAN_CPU(-), PEAK_CPU(-)	0.5, 0.5

Table 21. Causal topics inferred with logistic regression based Rubin Causal Model as base learner.

Learning Procedure	Theoretical Complexity	Approximate Complexity
BASE		
Bagging Augmented ElasticNet	$O(N(K^3 + K^2M)I)$	$O(NM)$
Signed Schweizer-Wolff Estimator	$O(NKM^2)$	$O(NM)$
AdaBoost-IPW Estimator	$O(NK^3MI)$	$O(NM)$
META	$O(WTNI)$	$O(N)$
ENSEMBLE	$O(T^2log(T)I)$	$O(T^2log(T))$

Table 22. Computational Complexity

causal effect estimation, producing multiple pairs of contradictory topics, such as C_{11} and C_{12}, and topics with reversed signs for correlated regressors such as C_3 and C_4.

G COMPLEXITY ANALYSIS

Table 22 presents the computational complexity of Hound's constituent methods with the following notation: **M**, **N** represent number of tasks per job and jobs per trace; **K**, **W**, **T** represent number of profiled metrics, words per document and topics to learn; **I** represents number of algorithmic iterations, a tunable parameter. We draw on prior complexity analyses. Bagging Augmented ElasticNet applies the Least Angle optimizer to fit coefficients [77]. AdaBoost-IPW estimator applies C4.5 to train weak learners [60]. Distributed online variational inference implements topic models[30]. Ensemble learning uses hierarchical agglomerative clustering [42].

In theory, Hound is in polynomial time—$O(\mathbf{NK^3M} + \mathbf{NKM^2})$. In practice, **K**, **T**, **W** are typically constants compared to **N** and **M**. For example, the Google and Spark traces report 21 and 37 task metrics (**K**), respectively. The number of output topics (**T**) is at most 15 and the number of words per topic **W** is at most 5. These parameters are much smaller than **N** and **M**. Given these practical considerations and a $1/\sqrt{\mathbf{M}}$ sampling rate for the SSW estimator, Hound's approximate complexity is $O(\mathbf{NM})$.

Received November 2017, revised January 2018, accepted March 2018.