PerfAugur: Robust Diagnostics for Performance Anomalies in Cloud Services

Sudip Roy*
Google Research
Mountain View, CA 94043, USA
sudipr@google.com

Arnd Christian König
Microsoft Research
Redmond, WA 98052, USA
chrisko@microsoft.com

Igor Dvorkin, Manish Kumar
Microsoft Corp.
Redmond, WA 98052, USA
{igord,maniku}@microsoft.com

***Abstract*—Cloud platforms involve multiple independently developed components, often executing on diverse hardware configurations and across multiple data centers. This complexity makes tracking various key performance indicators (KPIs) and manual diagnosing of anomalies in system behavior both difficult and expensive. In this paper, we describe PerfAugur, an automated system for mining service logs to identify anomalies and help formulate data-driven hypotheses. PerfAugur includes a suite of efficient mining algorithms for detecting significant anomalies in system behavior, along with potential explanations for such anomalies, without the need for an explicit supervision signal. We perform extensive experimental evaluation using both synthetic and real-life data sets, and present detailed case studies showing the impact of this technology on operations of the Windows Azure Service.**

I. Introduction

Cloud platforms need to constantly improve their quality of service to meet increasingly stringent performance and availability requirements from customers. For these, the quality life-cycle involves tracking various KPIs quantifying both performance and availability, and detecting and diagnosing any behavioral changes to the underlying system components.

While such detection and diagnosis is a non-trivial task for any system, the heterogeneity and complexity of cloud services makes this task particularly challenging. Cloud services continually deploy new software versions of system components and are executed on a diverse set of hardware configurations, often across multiple data centers. This heterogeneity and complexity creates both outright service failures as well as various *regressions* in performance/availability which manifest themselves in increased latency and/or increased failure rates of individual requests. While outright service failures are typically rare and quickly detected by operations teams, these types of performance/availability anomalies (which have also been named *chronics* [1] or *latent faults* [2]) can remain undetected for much longer periods as they typically only affect a subset of requests and occur under complex sets of conditions (which was also observed in [1]). Such anomalies may lead to service level agreement violations and thereby have significant financial implications for cloud service providers.

Anomaly diagnosis is made more difficult by the very skewed performance counter distributions typically seen in web services even for well-configured systems. Such skew was also observed in [3] and [4], especially for network round-trip times, where the 99th percentile time can be orders of magnitude larger than the median [5], [6]. Because of this skew, anomalies typically cannot be diagnosed using only a few instances of latent operations, since it is difficult to distinguish *transient anomalies* (aberrations in system behavior which occur under a unique and unrepeatable set of circumstances) from *systemic anomalies* (which are repeatable and unlikely to be resolved without developer intervention).

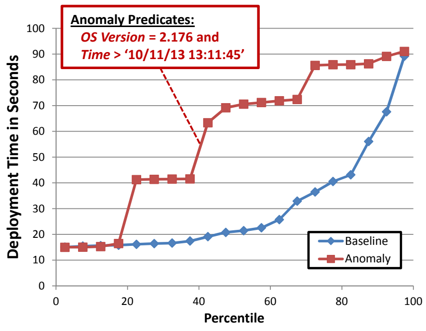


Fig. 1: An anomaly in VM deployment time detected by PerfAugur.

Manual diagnosis of regressions involves multiple iterations of analyzing service logs, formulating hypotheses based on experience, and finally validating the hypotheses against larger sets of data. This manual approach to diagnosis is not only slow but can also be expensive. Diagnosing these types of regressions would greatly benefit from an automated approach capable of identifying *both* (a) significant changes in system behavior relative to a baseline, and (b) system configurations which are correlated with the observed changes. Automating the first aspect allows subsequent ranking of various anomalies by importance as well as an initial assessment of the impact of fixing the root cause (i.e., deciding "*Which problem should I fix first?*"). Automating the second assists with the formulation of root-cause hypotheses and eliminates possible user bias towards their own (potentially incorrect) intuition.

Example 1.1: Consider a scenario where a code-change results in a significant increase in the duration of virtual machine (VM) deployments. Based on the service logs, PerfAugur not only detects this change, but also identifies that it is most pronounced for the instances with OS version 2.176 started after '10/11/13 13:11:45' (see Figure 1). With this knowledge, the developer can then investigate the code-changes relevant to OS version 2.176 around the specific time point.

*Work done during internship at Microsoft Research.

Problem Statement: We shall refer to these types of constraints on OS version and time as *predicates* that identify an anomalous subset of data. The problem we study in this paper can be formulated as computing compact sets of predicates that give rise to the most significant changes in behavior over a baseline.

A. Challenges

Absence of Supervision Signal: Unlike outright failures (which can often be identified from system logs), there is – in the cases of systemic anomalies we observed in practice – no simple and reliable way to identify the individual instances that make up the anomaly up-front. This make techniques that discover correlated predicates based on pre-labeled *failure cases* or outliers (such as e.g, DRACO [1], *Distalyzer* [7], or supervised anomaly-detection [8]) unsuitable for our scenario.

In the presence of the very skewed performance-counter distributions we discussed earlier, systemic anomalies often do not manifest themselves in the form of many additional outlier data points, but instead result in a noticeable increase in requests with moderately larger performance measures. A real-life example of this is our first case study (see Section VI), where a storage misconfiguration results in significant changes in median latency, but very little change in the 95th percentile (see Figure 10, left graph). As a result, the points making up a systemic anomaly can often not be reliably identified using one of the existing outlier detection techniques [8] up-front. *Two-step approaches*, in which we first identify the instances making up an anomaly using outlier detection techniques and subsequently use these outliers as supervision data to identify correlated predicates, are thus likely to perform poorly in these scenarios. We substantiate this experimentally in Section V-A3.

Robustness against Outliers: As opposed to the *two step approach*, we propose an approach that discovers predicates and performance changes jointly, by searching over the space of predicate combinations and comparing the difference between the distribution induced by a set of predicates and the overall baseline distribution.

Note that this approach also needs to account for the high data skew when comparing distributions: for example, using simple statistics such as *averages* for these comparisons can be problematic as they can easily be 'dominated' by a few outlier values. Instead, our approach is based on *robust statistics* [9] such as percentiles, which avoid this issue, and have become common industrial practice in Service Intelligence. Here, we define being robust to outliers using the notion of the *breakdown point* of an estimator [9]; intuitively, the breakdown point of an estimator is the proportion of incorrect observations (e.g., arbitrarily large deviations) an estimator can handle before giving an incorrect result.

Example "breakdown point": Given m independent variables and their realizations $x_1, \dots x_m$, we can use $\bar{x} = (\sum_{i=1}^{m} x_i)/m$ to estimate their mean. However, this estimator has a breakdown point of 0, because we can make $\bar{x}$ arbitrarily large just by changing any *one* of the x_i. In contrast, the median of a distribution has a breakdown point of 50%, which is the highest breakdown point possible.

Computational Overhead: There are primarily two sources of computational complexity in mining such anomalies robustly. First, robust statistics tend to be more expensive computationally, especially with respect to incremental computation. To illustrate this, consider the case of updating the mean of a distribution $D = \{x_1, \dots x_m\}$ when adding t additional points $\{x_{m+1}, \dots x_{m+t}\}$; the mean can easily be updated once the sum of the new values $sv = \sum_{i=1}^{t} x_{m+i}$ (in addition to m and t) is known. In contrast, updating the median requires knowledge of both the exact values of $\{x_{m+1}, \dots x_{m+t}\}$ as well as at least t additional values in D. As a consequence, many of the standard pruning "tricks" used in data mining (such as the optimizations discussed in [10] for a similar scenario) do not apply. Instead, different types of optimizations which leverage the specific characteristics of robust aggregates and the skewed distributions we see in practice are needed.

Second, testing all possible combinations of predicates for anomalies leads to an inherent combinatorial explosion. Given the size of underlying search space, the challenge is to develop algorithms which strike a balance between the quality of the mining result and computation cost.

B. Contributions

In this paper, we describe *PerfAugur*, a system for mining service telemetry information to identify systemic anomalies quickly and help formulate data-driven hypotheses as to the involved components and root causes. In particular,

- We formally define the problem of mining systemic anomalies based on robust statistics and propose algorithms that address the challenges imposed by the size of the search space and the computational overhead of maintaining robust statistics.
- We evaluate the resulting detection accuracy as well as overhead using various data sets and compare our approach to two well-known existing techniques, SCORPION [10] and decision-tree based diagnostics [11].
- Finally, we demonstrate the impact of our system on industrial practice via two real-life case studies of investigations into Windows Azure telemetry.

Note that the approach described in this paper generalizes to analytics tasks in other domains for which robust statistics are required due to data skew/noise.

II. RELATED WORK

Outlier Detection: There exists a rich body of (unsupervised) outlier detection techniques in data analytics literature (see [8] for an overview). However, the problem formulation studied in this paper is inherently different. We do not seek to identify all sufficiently anomalous *outlier* data points; instead we identify a compact set of predicates that induce a significant change in behavior over the baseline. Note that for such change to be significant, it does not require a large number of extreme values in the performance counters of interest. As a result, *two-step approaches* that first use outlier detection techniques to identify individual outlier data points and subsequently seek to identify predicates from these often fail to capture interesting systemic issues with system performance.

Log-based Diagnostics tools: Recently, there has been a large body of analysis tools that use machine-learning based approaches for the analysis of performance logs, including *Distalyzer* [7], the work of Cohen et al. [12] (which uses *Tree-augmented Naive Bayesian Networks* to identify combinations of metrics and threshold values that correlate with performance states (e.g., "*SLO violated*")), the work of Chen et al. [11] (which uses decision trees to mine paths that distinguish successful from failed executions), and *Fa* [13] (which uses *anomaly-based clustering* which clusters monitoring data based on how much it differs from the failure data), and *Draco* [1]. However, all of these techniques require a supervision signal in form of failure data or a set of abnormal instances in a separate log. This makes them unsuitable for our problem setting and the type of exploratory analysis PerfAugur is aimed at. In addition, most of the proposed techniques are incapable of handling the range of data types and scoring functions (used to quantify the significance of the change in behavior over the baseline) that PerfAugur is able to deal with.

Clustering based mappings for known anomalies: There is also been recent work to use *clustering*-based methods [13] or specialized *fingerprints* [14] to map current performance anomalies to one of a number of known failure states. Unlike our work, these techniques are optimized for accurate and fast matching and require signatures of known failures to be effective; in case of unknown, novel failures, they, unlike PerfAugur, provide little insight into potential root causes.

Other approaches: Two recent approaches, SCORPION [10] and the work of Roy et al. [15] propose techniques that "explain" outliers in aggregate queries by identifying subsets of tuples that contribute most to the outlier aggregate. While the approaches of both these papers are similar to PerfAugur as both try to identify some form of predicates which explain anomalous aggregate values, PerfAugur is targeted at performance diagnostics and, as a result, uses specialized classes of aggregate functions based on robust statistics. The predicate-search algorithms in PerfAugur are designed for robust statistics specifically, where none of the optimizations proposed in [10] and [15] apply. We experimentally compare the performance and result quality of SCORPION to our techniques in Section V.

A very different approach to "explaining" outliers in data is described by Micenková et al. [16]. Here, the explanations consist of the combinations of dimensions in which the outlier shows the largest deviations; these explanations are very different and far less intuitive in the context of diagnostics. Furthermore, it is not clear how to adapt these techniques for the various categorical data types handled by our approach. Müller et al. [17] describe a related approach that first identifies *relevant* subspaces using statistical significance tests and then uses these to rank outliers.

Aproximate Quantile Computation: There exist a number of techniques for the approximate quantile computation (typically, in the context of data streams), e.g. [18], [19]. The algorithms in this paper incorporate the computation of quantiles for all prefixes of an interval as a sub-step; PerfAugur computes these exactly, by maintaining a min- and a max-heap of the appropriate size (see [20]). Replacing this component of PerfAugur with one of the approximate techniques would reduce both the computational overhead (typically, by a logarithmic factor) and the memory requirements.

Time	VM Type	DataCenter	Latency
2014-01-19 03:14:17	IaaS	CA	30 ms.
2014-01-19 03:15:09	PaaS	NY	40 ms.
2014-01-19 03:15:57	PaaS	CA	43 ms.
2014-01-19 03:16:07	PaaS	CA	60 ms.

TABLE I: An example of a log relation.

III. ROBUST DIAGNOSTICS

Most cloud services use some form of measurement infrastructure that collects and compiles telemetry information in a suitable form for further analysis. For simplicity we assume that the telemetry information is maintained in a single relation R with attributes $A_1, ..., A_k$. Each tuple in this relation corresponds to a single measurement of a particular action. The set of attributes can be partitioned into two non-overlapping sets $\mathcal{A}_e$ and $\mathcal{A}_m$ such that $\mathcal{A}_e$ contains the set of attributes which describe the system environment under which actions are taken, and $\mathcal{A}_m$ contains the set of attributes each corresponding to a performance indicator. An example of such a relation is shown in Table I. Each tuple in this relation contains information pertaining to spawning a new virtual machine. For this relation the set $\mathcal{A}_e$ consists of the attributes *timestamp*, *virtual machine type* and the *data center location* and the set $\mathcal{A}_m$ simply contains the *latency* attribute.

Anomalies: Let $\Sigma(R, A_i)$ be some statistical property computed over values of the attribute A_i for all tuples in the relation R (e.g., a median). Given such a statistical property over a particular attribute $A_i \in \mathcal{A}_m$, an anomaly is a subset of the measurements $S \subseteq R$ such that $\Sigma(S, A_i)$ differs *significantly* from the baseline property defined by $\Sigma(B, A_i)$ over a baseline set B. In the absence of a pre-specified set B, we use $\Sigma(R, A_i)$ as the baseline measure.

Predicates: In this paper, we define predicates (denoted θ) as conjunctions of equality predicates of the form $A_e = v$ or range predicates of the form $v_{low} \prec A_e \prec v_{high}$, where $A_e \in \mathcal{A}_e$, v, v_{low} and v_{high} are constants, and $\prec$ defines a total order over the domain of the attribute A_e. Such predicates effectively summarize the system environment under which the anomaly occurs and therefore, characterize the conditions which may be related to the cause of the anomaly. In the rest of the paper, we refer to an environment attribute participating in a predicate as a *pivot* attribute.

Robust Aggregates: For any subset $S = \sigma_\theta(R)$, where σ is the relational selection operator, we can define how much S differs from R with respect to one specific performance indicator $A_m \in \mathcal{A}_m$ using suitable aggregate functions. For the reasons outlined in Section I-A, we only consider functions that are robust (denoted by Σ_r) to the effect of outliers in this context, such as the *median* or other *percentiles*.

Scoring Functions: We use the robust aggregates as part of so-called *scoring functions* to quantify the impact of an anomaly S with respect to an underlying *baseline* distribution. We use R as the baseline set in the following for simplicity; however, our approach works identically when the baseline is specified separately (e.g., as last month's measurements). We measure impact in terms of the change in distribution between S and R for a given performance indicator attribute A_m. A scoring function takes these three parameters (R, S, A_m) as input and outputs a single number used for ranking anomalies.

Each scoring function has to quantify two aspects of impact: (a) how different is the anomaly in terms of the change in (the distribution of) A_m, and (b) how many instances of operation/objects are affected by the anomaly. Note that two these factors trade off against each other: typically, the more points are included in an anomaly, the smaller is the change in distribution, and vice versa. Obviously, an anomaly covering all points in R would in turn have the baseline distribution and thus show no change at all.

To quantify the deviation in A_m, we use a robust aggregation function Σ_r to compute aggregates for the attribute A_m over all items in S as well as those in the baseline R. Subsequently, we measure the degree of the anomaly as the difference between these two values; we denote this difference using the notation $\Sigma_r(S, A_m) \sim \Sigma_r(R, A_m)$. Note that the choice of Σ_r as well as appropriate difference operator $\sim$ depends on the scenario and the type of the attribute of interest. When A is of a numeric type, Σ_r is typically a percentile and $\sim$ the absolute difference between the corresponding percentiles. On the other hand, for non-numeric categorical attributes (such as error codes or the names of failing function calls), we use the *KL-Divergence* [21] – a standard measure of distance between probability distributions. Here, the divergence is computed between the probability distribution of values of A_m in the baseline set (R) and the anomalous subset (S). Note that the KL-Divergence is a robust measure by default, as each individual item can not change the overall probability distribution disproportionately. We will discuss a real-life example of categorical attributes of interest in Section VI.

To quantify how many instances of operations or objects are affected by the anomaly we use a function of the size of S. In practice, we typically use the natural logarithm of $|S|$, giving us the following scoring function:

$$f(R, S, A_m) := \underbrace{\left(\Sigma_r(S, A_m) \sim \Sigma_r(R, A_m)\right)}_{\text{Deviation from baseline}} \times \underbrace{\log |S|}_{\text{Impact of \# instances}} .$$

Here, the use of the logarithm of the size of S (as opposed to using $|S|$ outright) favors anomalies that result in a larger deviation from the baseline (but over smaller number of instances). Note that the algorithms discussed in Section IV are also applicable when other functions of $|S|$ are used to quantify the effect of the number of instances after some suitable straight-forward modifications.

Diversity: Finally, in order to avoid providing multiple similar explanations for same anomalies or multiple explanations for the same set of anomalous measurements, we need to incorporate a notion of diversity into the mining task. For instance, the two predicates $v_{low} \prec A_e \prec v_{high}$ and $v'_{low} \prec A_e \prec v'_{high}$, such that $v_{low} \approx v'_{low}$ and $v_{high} \approx v'_{high}$, while different, convey almost identical information. Presenting both to the user is unlikely to convey any additional information. To incorporate this notion of diversity, our framework supports the specification of a diversity function $f_{div}(\theta_1, \theta_2) \rightarrow \{\texttt{true}, \texttt{false}\}$ which returns `true` if the anomalies explained by the predicates θ_1 and θ_2 are diverse, and false otherwise. The mining algorithms proposed in Section IV are independent of any specific diversity function.

While the exact definition of diversity may be user defined, below we propose a simple and meaningful diversity function that we will use in this paper going forward. Consider two atomic predicates, θ_1 and θ_2, defined over the same environment attribute A_e. As explained earlier, the notion of diversity must capture the degree of overlap between the two predicates. While there are multiple metrics to measure such overlap such as the Jaccard-distance between $\sigma_{\theta_1}(R)$ and $\sigma_{\theta_2}(R)$, an extreme form of diversity is to disallow any overlap, i.e., $\sigma_{\theta_1}(R) \cap \sigma_{\theta_2}(R) = \emptyset$. In this paper, we assume this as the default notion of diversity for atomic predicates.

The same principle can be extrapolated to anomalies defined by a conjunction of many atomic predicates. For such multi-predicate anomalies, it is likely that only a subset of the predicates also induce a relatively high-scoring anomaly. Consider the following example: if all deployments using build version 2.17 have abnormally high latency, then it is likely that the subset of deployments that use build version 2.17 *and* are deployed on cluster *XYZ* will also show high latencies. Therefore, unless the latency spike is specific to cluster *XYZ*, presenting an anomaly [$Build = 2.17 \wedge Cluster = XYZ$] in addition to the original anomaly [$Build = 2.17$] does not convey additional information and should be avoided.

Generalizing from the above, we define our default notion of diversity to multi-atom predicates as follows. Let $\mathcal{A}_\theta \subseteq \mathcal{A}_e$ be the set of environment attributes over which the atomic predicates of θ are defined. We consider two explanation predicates θ_1 and θ_2 diverse, if and only if, either $A_{\theta_1} \nsubseteq A_{\theta_2}$ and $A_{\theta_2} \nsubseteq A_{\theta_1}$ or, $A_{\theta_1} \subseteq A_{\theta_2}$ or $A_{\theta_2} \subseteq A_{\theta_1}$ and $\sigma_{\theta_1}(R) \cap \sigma_{\theta_2}(R) = \emptyset$. Intuitively, the first condition requires each of the explanations to have at least one distinguishing attribute. The second condition applies when the first condition does not, and similar to the atomic predicate case, requires them to explain non-overlapping set of measurements.

Problem Definition: Using the intuition introduced above, we can now define task of mining a set of diverse anomalies:

Problem 3.1 (Diverse Anomaly Mining): Given a telemetry log R, a particular measurement attribute $A_m \in \mathcal{A}_m$, the required number of anomalies k, a scoring function for Σ_r, $f(S, R, A_m) \rightarrow \mathbb{R}$ (denoted $f(S)$ in short) and a diversity function over explanations, $f_{div}(\theta_1, \theta_2) \rightarrow \{\texttt{true}, \texttt{false}\}$, the diverse anomaly mining task is to determine an ordered list of explanations, $L = [\theta_1, ..., \theta_k]$, which satisfy the following properties:

- *For any $1 \leq i < j \leq k$, $f_{div}(\theta_i, \theta_j)$ is* `true`*.*
- *For any $i < j$, $f(\sigma_{\theta_i}(R)) > f(\sigma_{\theta_j}(R))$*
- *Finally, for any $\theta_i \in L$ and $\theta \notin L$, either $f(\sigma_{\theta_i}(R)) > f(\sigma_\theta(R))$ or there exists $j < i$ such that $f(\sigma_{\theta_i}(R)) < f(\sigma_\theta(R)) < f(\sigma_{\theta_j}(R))$ and $f_{div}(\theta_j, \theta)$ is* `false`*.*

Informally, the first condition requires all the top-k predicates to be diverse. The second condition ensures that the list of predicates is ordered according to their scores. Finally, the third condition ensures that a high-scoring predicate is excluded from the top-k anomalies only if there exists an higher scoring predicate with respect to which the predicate is not diverse.

IV. Anomaly Mining Algorithms

In this section, we propose algorithms for the diverse anomaly mining task. Our algorithms extract predicates which identify the top-k highest-scoring diverse anomalies for a measurement log R. First, we describe algorithms for identifying anomalies defined by atomic predicates over a single attribute in $\mathcal{A}_e$, called the *pivot* attribute. Subsequently, we describe algorithms for anomalies with multiple pivot attributes.

The particular algorithm used for mining anomalies depends on the type of pivot attribute. We refer to pivot attributes which have an inherent order over values such as numerical and date-time datatypes as *ordered pivots*. On the other hand, attributes which enumerate values from a certain domain like cluster names and OS versions are called *categorical pivots*. While for ordered pivots we extract range predicates of the form $v_{low} \prec A_e \prec v_{high}$, for categorical pivots, we extract equality predicates of the form $A_e = v$, where A_e is the pivot attribute. Identifying anomalies for categorical pivot attributes is computationally easy since the problem reduces to performing a group by over the pivot attribute followed by computing each group's score. Therefore, for the rest of the section we specifically focus on algorithms for ordered pivots.

Notation: We use A_m to denote the performance indicator over which anomalies are to be detected, A_e to denote a pivot attribute and θ_{ij} as a notational shorthand for the range predicate $v_i \prec A_e \prec v_j$, where v_i and v_j are the i^{th} and j^{th} values of the pivot attribute in sorted order. We also use S_θ as a notational shorthand for $\sigma_\theta(R)$.

A. Single Pivot Anomalies

We first propose an exhaustive algorithm for ordered pivots. However, this brute force approach does not scale well to very large data sets. To overcome this, we then propose two additional algorithms: first, a order-of-magnitude faster algorithm that extracts predicates such that the anomaly scores are (at least) within a constant factor, α, of those mined exhaustively. Second, an even faster algorithm which performs well in practice and offers a performance guarantee which depends on the data characteristics itself.

1) Naïve: The exhaustive algorithm for identifying anomalies on ordered pivots sorts the items by the pivot attribute, and then scores the subset of items within every pair of start and end-points. The computational complexity of this algorithm depends on the cost of computing the scoring function. For a median-based scoring function, this cost is $O(|\sigma_\theta(R)|)$, where θ explains the anomaly being scored. However, the cost of determining the median for an interval $\theta_{i(j+1)}$ given the median for θ_{ij} can be reduced to $O(\log|\sigma_{\theta_{ij}}(R)|)$, by maintaining the medians of the interval incrementally with two heaps [20] – a max-heap and a min-heap; this approach also works for all other percentiles, all that changes is the fraction of tuples in each heap. Given this incremental implementation of the scoring function, the cost of the exhaustive algorithm (for $N = |R|$ items) becomes $O(N^2 \log N)$.

2) Grid-Refinement: Next, we propose an algorithm which offers a principled way to trade off the "accuracy" of the mined anomalies for efficiency; instead of returning the highest-scoring anomaly, the algorithm returns an anomaly whose score is guaranteed to be within a factor α of the highest-scoring anomaly (where e.g., $\alpha = 0.9$). In return for relaxing the score constraint, this algorithm performs orders of magnitude faster in practice. The speedup seen by this algorithm is the result of exploiting two properties typically found in data distributions seen in the context of cloud diagnostics:

```
$\mathcal{Q} \leftarrow \emptyset$ {A priority queue of anomalies sorted by an upper bound on their score.}
Let $N = |R|$
$R_s = Sort(R, A_e)$ {Sort instances by pivot attribute $A_e$}
$\mathcal{Q}.push(\theta_{1N}, 0, \infty, N)$ {Initialize $\mathcal{Q}$}
$TopK \leftarrow \emptyset$ {The result set.}
while $\mathcal{Q} \neq \emptyset \wedge |TopK| < k$ do
    $(\theta, s, u, g) \leftarrow Q.dequeue$
    if $s/u \geq \alpha$ then
        if $\bigwedge_{\theta_i \in TopK}(f_{div}(\theta, \theta_i))$ then
            $TopK.Add(\theta)$
    else
        for all $r \in Refine(\theta, g)$ do
            $\mathcal{Q}.push(r)$
return $TopK$
```

Algorithm 1: The α-approximate grid-refinement algorithm.

"Small" Anomalies: First, for most datasets, anomalies are expected to constitute a relatively small fraction of all the items. The naïve algorithm spends significant amount of computation time in ruling out intervals which resemble the baseline and are, therefore, non-anomalous. In contrast, the new algorithm can rule out large fractions of the search space quickly by bounding the score of the anomalies in it.

Stability of Robust Statistics: For the data distributions typically seen in practice, robust statistics are relatively stable with respect to the addition or removal of a small number of points. To illustrate this, consider Figure 2 which shows an example latency distribution, and the corresponding median. Here, the middle portion of this distribution tends to be "flat", implying that the median does not change significantly in response to the insertion or deletion of k points (which can at most move the median by k points along the x-axis, corresponding to only a small change along the y-axis). This property of stability implies that the score of an anomaly $v_{low} \prec A_e \prec v_{high}$ is expected to be approximately equal to that of an anomaly defined by $v'_{low} \prec A_e \prec v'_{high}$ if $v_{low} \approx v'_{low}$ and $v_{high} \approx v'_{high}$. The algorithm exploits this by using the score of one anomaly to compute tight upper bounds on the scores of anomalies with similar predicates.

Intuitively, the algorithm uses grids of various levels of coarseness to "zoom into" regions in the data containing high-scoring anomalies. First, the algorithm analyzes the data at a coarse granularity, choosing the values of v_{low} and v_{high} only from the points along the grid and computing upper bounds on the possible scores of anomalies found at finer granularity. Only for sub-regions where these upper bounds are sufficiently high, do we then consider anomalies found at a finer grid resolution, repeating the process until we discover an anomaly with a score within a factor of α of the highest (potential) score of all unseen anomalies. We illustrate this process in Figure 3.

Refinement and Bounding: The *Grid-refinement* algorithm

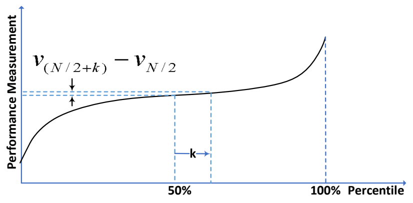


Fig. 2: Small change in median value due to addition of k points below the median.

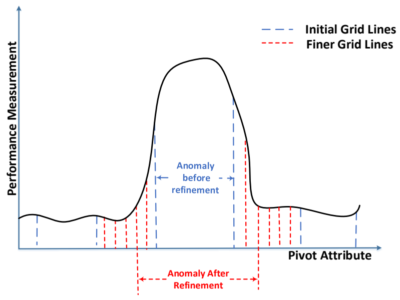


Fig. 3: Refinement by searching the grid near interval endpoints at a finer resolution.

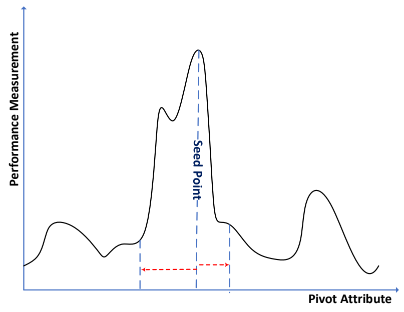


Fig. 4: Expansion from a *seed* point to identify anomalies.

```
Let $\theta_{low,high}$ at grid size $g$ be the interval to be refined.
$g_r \leftarrow g/$ConvergenceRatio {Refined grid size.}
$\mathcal{Q}_{refined} \leftarrow \emptyset$ {The set of refined anomalies.}
for $i \leftarrow (low - g) : g_r : low$ do
    for $j \leftarrow (i + g_r) : g_r : (high + g)$ do
        $s_{ij} = f(R, S_{\theta_{ij}}, A_m)$
        $u_{ij} = BoundScore(R, S_{\theta_{ij}}, A_m, g_r)$
        $\mathcal{Q}_{refined}.Add(\theta_{ij}, s_{ij}, u_{ij}, g_r)$
return $R$
```

Algorithm 2: Refinement procedure for a predicate $\theta_{low,high}$ at grid size g. Boundary-condition checks including handling of intervals smaller than or equal to a grid size are omitted for simplicity.

is shown in Algorithm 1. The algorithm maintains a priority queue of anomalies represented by 4-tuples (θ_{ij}, s, u, g), where θ_{ij} is the interval, s is the score of the current interval, u is an upper bound on the score achievable through arbitrary refinement of the grid near the end-points of the interval $[v_i, v_j]$, and g is the current grid size. The algorithm dequeues anomalies from the priority queue in order of their upper bound on scores; if the current score is within an α factor of the bound on the scores, then we add it to the result set after checking the diversity constraint. Otherwise, the interval is refined using the "zoom in" procedure shown in Algorithm 2. During refinement of an interval, for each possible refined interval at a finer grid size, we compute the score of the anomaly as well as an upper bound on the possible improvement achievable by "refining" the grid, i.e., the maximum score possible for an anomaly when using (a) an arbitrarily fine grid and (b) the endpoints v_{low} and v_{high} being within one grid size of the original "coarse" pair of endpoints (see Figure 3). The algorithm terminates once the top-k approximate anomalies are determined.

Bounding Scores: For correctness we require the function $BoundScore$ to provide a *sound* upper bound on the score of any given predicate, i.e., for any interval θ_{ij} at grid g, if $\mathcal{Q}_{refined}$ is the set of intervals obtained by refining θ_{ij} as shown in Algorithm 2, then $\forall \theta_{i'j'} \in \mathcal{Q}_{refined}, f(S_{\theta_{i'j'}}, R, A_m) < u$. We show one such method of estimating the upper bound for scoring functions using the median as the robust statistic of choice. Extending it to arbitrary percentiles is trivial using a similar technique.

Let $S_{\theta_{ij}}$ be an interval at grid size g for which the upper bound is to be estimated. The specific method of refinement shown in Algorithm 2 restricts the maximum deviation of the median to $2g$ points since the refinement only allows addition of points by expansion of the interval by at max g points on either end of the interval. Let v_k be the k^{th} value in sorted order of the attribute A_m among the points in $S_{\theta_{ij}}$. Therefore, $v_{N/2}$ denotes the median value. Since the median for any refinement can at most deviate from the median by $2g$ points, the score for any refinement of the interval is bounded by $(v_{N/2+2g} - v_{N/2}) \times \log(|S_{\theta_{ij}}|)$. For typical distributions, the change in median value, and therefore the gap between the upper bounds and the (best) actual score for an interval, is expected to be relatively small due to the stability around medians illustrated in Figure 2. Note that both refinement and computing an upper bound on scores for intervals of size equal to one grid block need to be handled specially; the details are omitted due to space limitations.

Correctness: The algorithm satisfies the invariant that an anomaly is added to the set of top-k anomalies if and only if the anomaly's score is within an α factor of the highest scoring anomaly. Let S_θ be the first anomaly to be included in the top-k by the algorithm shown in Algorithm 1. Also, let S_θ^{opt} be the highest scoring anomaly. Also, let S_θ be an anomaly at a grid resolution of g. Let S_β be the anomaly which contains S_θ^{opt} and has both endpoints at the grid with resolution g. Since the algorithm dequeues anomalies according to upper bounds on scores, we know that $u(S_\theta) \geq u(S_\beta)$. By soundness of the bounding function and the refinement procedure, we also know that $u(S_\beta) \geq f(S_\theta^{opt}, R, A_m)$. Therefore, $u(S_\theta) \geq f(S_\theta^{opt}, R, A_m)$. Also, since the algorithm chooses the anomaly, we know that $f(S_\theta, R, A_m)/u(S_\theta) \geq \alpha$. Therefore, $f(S_\theta, R, A_m) \geq \alpha \times f(S_\theta^{opt}, R, A_m)$.

3) Seed Expansion: The *Grid-Refinement* algorithm relies on the stability of medians property (see Figure 2). However, the distributions seen around much higher (or much lower) percentiles are often less stable. We now propose an algorithm for faster detection of anomalies aimed in particular at scoring functions based on these percentiles, or for fast analysis of very large data sets. This algorithm offers a significantly lower asymptotic overhead ($O(N^{1.5})$) as well as significantly faster wall-clock runtime. However, as opposed to *Grid-Refinement*, which guarantees a constant approximation ratio, the scores of the anomalies mined by the Seed Expansion algorithm are within a data-dependent factor of the optimal anomalies.

The intuition behind this algorithm is based on anomalies for high/low percentiles typically containing extreme (i.e., relatively high or low) values for the performance indicators; to

```
Let $s$ be the index of the seed in sorted order of pivot $A_e$
$l_{new} \leftarrow s$; $r_{new} \leftarrow s$
$MaxScore \leftarrow -\infty$
while $f(S_{[l_{new},r_{new}]}, R, A_m) \geq MaxScore$ do
    $l \leftarrow l_{new}$; $r \leftarrow r_{new}$
    $MaxScore \leftarrow f(S_{[l_{new},r_{new}]}, R, A_m)$
    $score_l \leftarrow f(S_{[l-1,r]}, R, A_m)$
    $score_r \leftarrow f(S_{[l,r+1]}, R, A_m)$
    $score_{lr} \leftarrow f(S_{[l-1,r+1]}, R, A_m)$
    Let $[l_{new}, r_{new}]$ be the interval corresponding to
    $\max(score_l, score_r, score_{lr})$.
return $[l, r]$
```

Algorithm 3: Expansion of a single seed point (l and r denote left and right, respectively).

simplify exposition, we will make the assumption that we only seek anomalies corresponding to large performance indicator values. The algorithm first chooses the top-$\sqrt{N}$ number of points in order of value of the performance indicator; we call these points *seed points*. For each seed point we now want to determine whether it corresponds to an isolated transient anomaly (which we want to ignore), or is part of a systemic anomaly (which we want to detect). In the former case, we expect the seed point to be a local extremum surrounded (along the pivot axis) by many points that roughly resemble the baseline distribution; in the latter case, we expect further extreme measurement values in the neighborhood of the seed.

Smoothing: To avoid situations where all the seed points chosen are transient anomalies, we apply an initial smoothing step before choosing the seed values. Here, we first replace each value v_i of the performance indicator with the median value among all values in an interval along the pivot-axis of size c and "centered" at v_i; we then chose the largest values among these. This way, single outlier points within a region of low values are not chosen as seeds, eliminating (single-point) transient anomalies from consideration.

Given any seed point identified by the index s with the pivot value v_s, the algorithm initializes a single-item anomaly with the predicate $v_{low} = v_s \prec A_e \prec v_{high} = v_s$ and tries to *expand* this anomaly by adding points in each direction along the pivot axis. If the seed point is part of a systemic anomaly, we expect the score of the resulting anomaly to grow with the expansion. On the other hand, if the seed corresponds to a transient anomaly, we expect the score to decrease (eventually) as points resembling the background distribution are added.

The procedure for expansion of a single seed point is shown in Algorithm 3. The algorithm expands a seed until an expansion does not result in an improvement in the anomaly score. This expansion procedure is repeatedly invoked for $\sqrt{N}$ seed points. Seed points which are already included in the expanded anomalies formed out of previous seed points are excluded from consideration as seeds. The algorithm maintains all expanded intervals in a sorted list from which the highest-scoring set of k diverse anomalies (Section III) is returned as the final result. The pseudocode for the complete algorithm is omitted due to space constraints.

Score Approximation: The quality of the anomalies mined by the seed expansion algorithm depends on how easily distinguishable the anomalies are from the background distribution. We use two properties of the dataset to quantify this distinctiveness of anomalies. First, the maximum gradient (i.e. $max_i(v_{i+1} - v_i)$) of the performance indicator attribute with respect to the pivot attribute, denoted δ_{max}. Note that this measure is computed *after* smoothing, effectively making this the maximum gradient over any interval of size c. Second, let $\Delta = \frac{v_N - v_{N/2}}{N/2}$ be the average gradient between the median and the maximum value. Also, let $\alpha = \frac{\delta_{max}}{\Delta}$. Then it can be shown that if S_θ is the best anomaly mined by the *Seed Expansion* algorithm and $S_{\theta_{opt}}$ is the top scoring pattern mined by an exhaustive algorithm, then $f(S_\theta, R, A_m) \geq \frac{2\log(N\alpha)}{\alpha \log N} f(S_{\theta_{opt}}, R, A_m)$, where f is the median-based scoring function and $|S_{\theta_{opt}}| \leq \sqrt{N}$. While we omit a formal proof due to space constraints, we briefly explain the implications of such a bound. For a distribution with a very pronounced anomaly, the value of α is expected to be high since δ_{max} is expected to be high. This in turn implies that the approximation factor $\frac{2\log(N\alpha)}{\alpha \log N}$ evaluates to a lower value since the contribution of α to the denominator dominates. Therefore, as expected, if anomalies are more pronounced in a distribution, the algorithm can identify the anomalies more accurately, giving us the desired behavior of identifying the most prevalent anomalies in a highly scalable manner.

B. Multi-Pivot Anomalies

Anomalies often occur due to system conditions which can only be reliably captured by predicates over multiple attributes. For example, response times for operations may degrade only under high memory contention when there also are multiple active threads on a machine. A brute force approach for identifying such multi-attribute anomalies would be to check all combinations of predicates for all subsets of environment attributes, which is clearly computationally prohibitive. This computational hardness is not unique to our problem, but is an instance of a general class of problems observed in other domains, such as optimal decision tree construction. Similar to well-known techniques used in the literature for similar contexts, our first approach is to construct multi-pivot anomalies greedily.

In practice, the vast majority of anomalies are detected well using greedy techniques, however, to detect anomalies that are not, we next propose an algorithm that co-refines pivots jointly across different attributes. Finally, we show how we can leverage a property typically seen in real-life data distributions (namely, a bound on the extent to which the score of the highest-scoring anomaly characterized by l predicates is reduced when we only consider a subset of the predicates) to provide a tractable algorithm that gives quality guarantees on the scores of the mined anomalies.

Since the greedy algorithm is straight-forward (and similar to traditional greedy decision tree construction algorithms), we omit the details of this algorithm.

1) Co-refinment: A purely greedy algorithm for mining anomalies may split a single anomaly into multiple anomalies due to lack of foresight into the potential refinement by joining with other pivots. For handling such corner cases, we propose a co-refinement strategy: here, we first run the greedy mining algorithm on a small random sample of the data with a weighted scoring function where each data point is weighted

```
$R^\gamma \leftarrow RandomSample(R, \gamma)$ {Choose a random sample w/o replacement of size $\gamma \times |R|$}
$f^\gamma(R^\gamma, S, A_m) := (\Sigma(R^\gamma, A_m) \sim \Sigma(S, A_m)) \times \log(\frac{|R^\gamma|}{\gamma})$
$TopKCoarse \leftarrow GreedyMine(R^\gamma, f^\gamma, A_m)$
$TopKRefined \leftarrow \emptyset$
for all $\theta_c \in TopKCoarse$ do
    $\theta_r \leftarrow \theta_c$; $g \leftarrow |\theta_r|$
    while $g >= 1$ do
        for all $\theta_r^i \in \theta_r$, where $\theta_r = \bigwedge_i \theta_r^i$ do
            $\theta'_r \leftarrow \theta'_r \wedge \arg\max_{\theta \in Refine(\theta_r^i, g)} f(S_\theta, S_{\theta_r}, A_m)$
        $\theta_r \leftarrow \theta'_r$;
        $g \leftarrow g/$ConvergenceRatio
    $TopKRefined.Add(\theta_r)$
return $TopKRefined$
```

Algorithm 4: A sampling and co-refinement based scheme for multi-pivot mining using a greedy mining procedure, $GreedyMine(R, f, A_m, k)$, which returns the top-k multi-pivot anomalies ordered by the scoring function f. We use θ_c to denote the predicates on the sampled data and θ_r to denote the predicates on the entire data.

by the inverse sampling ratio. This gives us an initial "rough" set of anomalies. We then co-refine these anomalies using the full data set as follows: we adopt an approach similar to the *Grid Refinement* algorithm of gradually "zooming in" to determine the exact interval boundaries for each predicate. However, instead of refining attributes one after the other, for each anomaly, we determine the best intervals across *all* constituent pivot attributes at a particular grid-size before drilling down to the next grid level (see Algorithm 4).

2) α-approximate Multi-Pivot Refinement: While computing the top-scoring anomalies for adversarial data distributions is computationally prohibitive, we can leverage properties typically seen in real-life data to obtain a tractable algorithm with absolute guarantees on the anomaly score. First, to illustrate these data properties, consider an example anomaly which is best characterized by intervals along two different pivot attributes. The heat-map representation of the anomalous measurement values w.r.t. to the two pivot attributes for such an anomaly is shown in Figure 5. The figure also shows three percentile distributions; two for (predicates on) each of pivot attributes when considered independently and a third for when they are considered together. Clearly, the deviation between the anomaly median and the background distribution, observed when both the attributes are considered together, shifts towards higher percentiles when only one of the pivots is considered. This is due to the addition of non-anomalous points to the anomaly. These non-anomalous points can only be filtered by pivoting on the secondary attribute. By limiting the extent to which this shift occurs, we can provide sound bounds for the improvement possible in anomaly scores. We formally state this assumption as follows:

Definition 4.1 (Maximum Refinement Ratio): Given a multi-pivot anomaly delimited by l predicates over pivot attributes, the maximum refinement ratio is the largest constant γ such that there exists an ordering of the predicates $\mathcal{O}$ such that $\frac{|S_{\wedge_{i+1}\theta_{\mathcal{O}(i+1)}}|}{|S_{\wedge_i \theta_{\mathcal{O}(i)}}|} \geq \gamma$, where $\gamma \in [0, 1]$.

Bounding Multi-Pivot Anomaly Scores: We assume that for a given log relation R and a performance indicator attribute A_m,

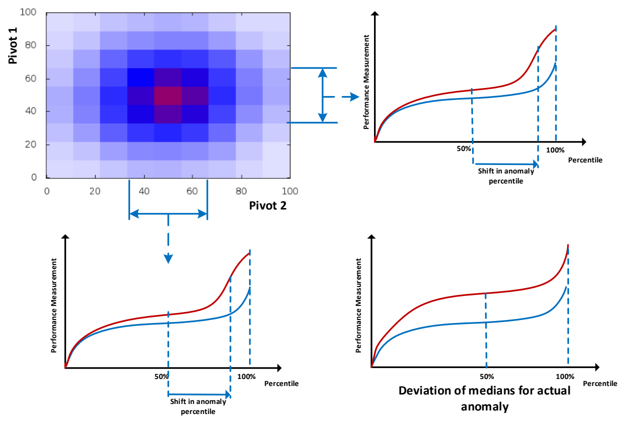


Fig. 5: Shift in the anomaly percentile on projection of anomaly over either of the two pivot attributes as compared to jointly over both.

the maximum refinement ratio γ is either known or is estimated conservatively ($\gamma = 1$ being most conservative). Under this assumption, given an l-pivot anomaly S_{θ_l}, it is possible to get an estimate of the potential improvement in the anomaly score by pivoting on additional attributes. Let $n = |S_{\theta_l}|$. If the maximum number of attributes in any anomaly is m, for any l-attribute anomaly, the minimum size of an m-predicate anomaly formed by extending S_{θ_l} has size at least $n_{min} = \gamma^{m-l} n$. For the particular case where the aggregation function is median, the maximum score obtainable by extending S_{θ_l} is then bounded by

$$\max_{i \in [n/2, n(1-\gamma^{m-l}/2)]} (v_i - v_{\lceil n/2 \rceil}) \times \log(2i).$$

This is because in the best case, all the points filtered by additional pivots are lower than the median value S_{θ_l}, and therefore cause a rightward shift of the median. As more predicates over pivots are added to the anomaly, this estimate becomes tighter.

As in the case of the single-pivot Grid Refinement algorithm, by maintaining an upper bound over the best possible l-pivot (unseen) refinements for anomalies with fewer pivots, we can design an α-approximate multi-pivot mining algorithm.

We omit the pseudo-code for the multi-pivot anomaly mining algorithm since it is structurally similar to the Grid Refinement algorithm (Algorithm 1), except for the refinement procedure and the initialization step.

V. Experimental Evaluation

In this section we evaluate the efficiency and accuracy of PerfAugur in comparison to Scorpion [10] and the decision-tree based approach of [11]. In addition, we test the feasibility of "two-step" approaches, which use outlier detection to provide the input labels for supervised techniques such as [7], [12], [11], [13], [1], by evaluating the accuracy of this labeling for real-life anomalies (Section V-A3).

Datasets: Since finding anomalies over ordered pivots is computationally significantly harder than for categorical pivots, we focus the experimental evaluation on ordered pivots. In contrast, the case studies in Section VI will illustrate examples with categorical pivots and both numerical as well as categorical performance indicators. For our experiments we use two types of data – actual log data and synthetically generated data. The actual log data is real telemetry data obtained from Windows Azure operations. For synthetic data, we adopt an approach similar to that of SCORPION. We create a relation R with d environment attributes and one measurement attribute as follows. We draw "normal" values of the measurement attribute from a distribution $\mathcal{N}(10, 10)$; then, to create d-dimensional anomalies, we choose a region of proportionate size in d-dimensional pivot space and assign measurement values drawn from $\mathcal{N}(80, 10)$ for all points in that space. Typically, these anomalies cover 10% of points. In order to inject multiple anomalies, we use multiple, non-overlapping copies of data generated as above.

Alternative Approaches: SCORPION generates predicates which, given a user identified set of outlier and holdout (to indicate the baseline) groups of observations, *explain* outlier observations in the data. The non-robust scoring functions used by SCORPION scores predicates by the extent to which they "influence" the outlier groups but not the holdout groups. The influence of a predicate is computed by measuring the difference in the aggregate values before and after removing the points satisfying a particular predicate. Because the various performance optimizations described in [10] don't apply to robust aggregates, we use the naïve implementation of Scorpion which performs an exhaustive search for such predicates. While the naïve implementation has a high computational overhead, it is guaranteed to produce the best quality predicates as per SCORPION's scoring function.

The original *decision-tree* based approach of [11] requires a supervision signal (in the form of *success/fail* labels), which is generally difficult to obtain in practice; here, however, we avoid such labeling by re-phrasing the underlying task as a *regression* problem, making the (numeric) measurement attribute the variable to be approximated by the tree and using a regression tree implementation based on [22] to compute the tree. This way, the tree will partition the space into regions with similar measurement values. We can now treat each partition in the data induced by the tree as a potential anomaly. We give this approach one additional "advantage" in that we set the number leafs in the tree (= number of partitions) to the number required to bound all anomalies in the synthetic data. Both SCORPION and the *decision-tree* approach have a number of tuning parameters; all our reported results are for the best configuration obtained after tuning both algorithms.

Setup: All experiments are run on a Intel Xeon L5640 processor with 96 GB of main memory running Windows 2008 R2 Enterprise. All reported values except for brute force algorithms are averages over more than 10 runs.

A. Experimental Results

For all our experiments, we compare the different algorithms in two respects – computational overhead, and the quality of the extracted anomalies. For quantifying quality, we use F_1-scores defined as $\frac{2 \cdot precision \cdot recall}{precision + recall}$ measured w.r.t. the known set of anomalous points injected synthetically. Since the ground truth is not known for the real logs, we use the ratio of scores of anomalies to those mined by *Naïve* as the quality metric. While PerfAugur supports scoring functions based on arbitrary percentiles, due to lack of space, we show experiments for scoring-functions based on medians only.

	Naïve	Seed	Grid (0.8, 0.9, 0.95)			SCOR.	DT
Synth.	0.95*	0.83	0.83	0.86	0.89	0.75	0.36
Real	1.0	0.77	0.98	0.98	0.99	-	-

TABLE II: Comparison of quality of anomalies mined over 100k items. For synthetic data where anomalies are known, we use F_1 scores. For real data, we use the ratio(SR) of the anomaly score to those mined by Naïve. *Reported values for Naïve are for 10k items, as computation for 100k items did not complete.

1) Experiments on One-Pivot Anomalies: In the first set of experiments, we study the behavior of the five algorithms (*Naïve, Grid-Refinement*, *Seed Expansion*, SCORPION, and decision-tree (DT)) for discovering top-scoring anomalies for a single (ordered) pivot under different data distributions; for the Grid-refinement algorithm, we also vary α, the approximation-ratio of the scores of the returned anomalies.

Scalability: For this experiment, we increase the number of items from 10k to 100k, introducing a new anomaly for every 10k items. Figure 6 shows a comparison of the overhead for mining the top anomaly with increasingly larger numbers of items. *Seed Expansion* and DT algorithms are about 3-4 orders-of-magnitude faster as compared to *Naïve* and SCORPION. Similarly, for any approximation ratio, the *Grid Refinement* algorithm is around two orders of magnitude faster than *Naïve* and SCORPION. The corresponding values of F_1-scores for 100k items are shown in Table II. In comparison to SCORPION and DT algorithms, all of our algorithms have significantly higher F_1-scores as shown in Table II.

Over real data, almost all of our algorithms are able to find an anomaly with almost the same score as those mined by *Naïve*. Since scoring metrics for SCORPION and DT are incomparable to our scoring function, it is not a fair comparison to evaluate their quality over real data using this approach.

Robustness: We next test the robustness of the algorithms as we add random noise simulating transient anomalies. We construct a synthetic dataset of size 4k containing a single anomaly. We then mine for the top-1 anomaly while varying the percentage of noisy points up to 30%. The overhead and quality results are shown in Figures 7 and 8.

While SCORPION and DT perform well in the absence of noise, their performance degrades rapidly as we add noise (Figure 8). In contrast, the *Grid Refinement*, *Seed Expansion* and *Naïve* algorithms, are able to tolerate up to 25% noise without significant degradation in quality. Moreover, the *Grid Refinement* algorithm requires more time as the percentage of noisy points increases to satisfy the guaranteed approximation ratio. Also, the higher the value of the approximation ratio α, the longer the time required to achieve it. In contrast, *Seed Expansion* is independent of the noise values as the number of seed points used is dependent on the data size only.

2) Experiments on Multi-Pivot Anomalies: In this set of experiments, we study the behavior of the four multi-pivot anomaly mining algorithms (*α-Approx*, *Greedy Multi-pivot*,

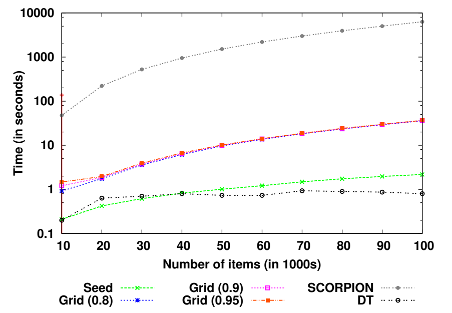


Fig. 6: Comparison of time required to mine the top-1 pattern on synthetic data. Times for Naïve are omitted as the algorithm takes more than a day over 100k items.

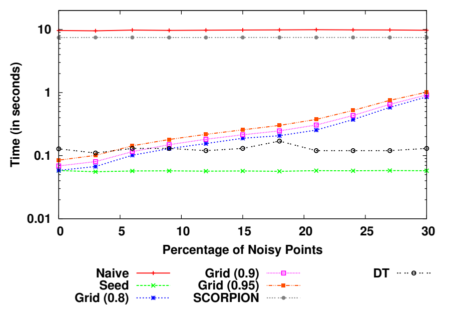


Fig. 7: Time for mining with increasing percentage of noise over 4k items.

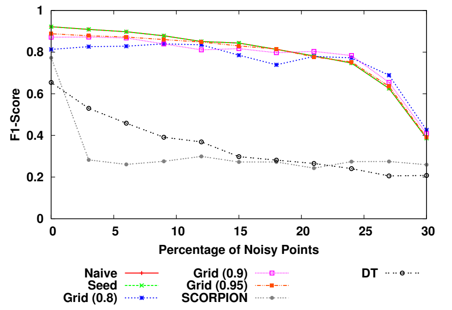
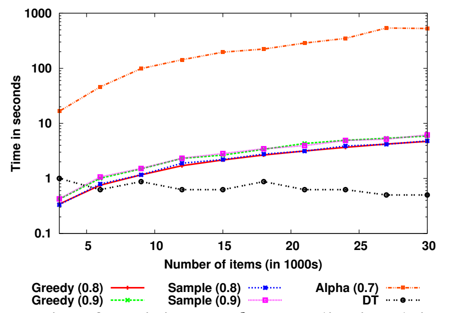
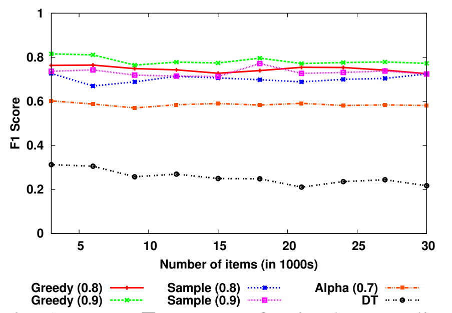


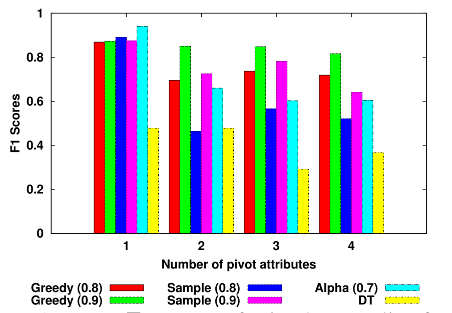
Fig. 8: F_1 scores with increasing percentage of noise over 4k items.



(a) Time for mining top-k anomalies involving 3 pivots with increasing number of items.



(b) Average F_1-scores of mined anomalies with increasing number of items.



(c) Average F_1-scores of mined anomalies for 10k items with increasing number of pivot attributes.

Fig. 9: Evaluation of multi-pivot mining algorithm.

Sample and Co-refinement (SC) and decision-tree (*DT*)). In our experiments, for α*-Approx*, we use $\alpha = 0.7$ and $\gamma = 0.7$. For *Greedy Multi-pivot* and *SC*, we use the Grid-refinement algorithm as an atomic predicate miner with α values set to 0.8 and 0.9. For *SC*, we use a sample size of 2%. Finally, since the Naïve implementation of SCORPION suffers from combinatorial explosion, we are unable to compare against SCORPION for multi-pivot anomalies.

To simulate transient anomalies, we add 20% noise values drawn from $\mathcal{N}(400, 10)$. Furthermore, in practice we have observed that for many data sets, a small fraction of performance measurements are disproportionately high (partially due to time-outs, instrumentation errors, etc.). To simulate these measurements, we sample 2% of the data (independently of whether these are part of the anomaly or not) and scale up their performance indicator by 100x.

Scalability: For this experiment, we use items with a single performance indicator and three pivot attributes. The anomaly is most precisely identifiable with three different predicates; one over each pivot attribute. We first study the behavior of the algorithms as we increase the number of items in the set from 3k to 30k. The data set contains 1 anomaly per 3k items. We evaluate the quality of the algorithms by comparing the average F_1-score of the top-k anomalies in the dataset. The results are as shown in Figures 9a and 9b.

While DT is the fastest of all algorithms, the F_1-score of the mined anomalies is quite low and it often misses some of the top-k anomalies all together. However, the *Greedy* and *SC* strategies scale well with the number of points and return all the anomalies as indicated by the high F_1 scores. Finally, the α-approx multi-pivot algorithm also has high values of F_1 scores but takes significantly longer to guarantee the desired approximation ratio. While the behavior of *Greedy* and *SC* are almost similar, it is possible to construct adversarial distributions in which *Greedy* performs poorly but, due to co-refinement, *SC* performs reasonably well. A recommended strategy is to use an ensemble of both. Due to lack of space, we are unable to present this experiment.

Varying the dimensionality: In this experiment, we study the behavior of algorithms with increasing dimensionality of the dataset. As explained in Figure 5, the higher the dimensionality, the more obscure the anomaly in its projection along any one pivot attribute, making it more difficult to detect.

For this experiment we keep the number of items in the dataset constant at 10k with a single anomaly. As shown in Figure 9c, with increasing number of pivot attributes, the quality of anomalies mined by DT decreases. However, for more than one dimension, the quality of the anomalies mined by each of the other algorithms is relatively unchanged; particularly so for the α-approx. algorithm. This can be attributed partly to the use of robust statistics and partly to the fact that even at higher dimensions, the projections of the attribute over any pivot attribute is still "anomalous" enough to be picked up by the greedy approaches.

3) Experiments on Two-Step Approaches: In this experiment, we tested the feasibility of “two-step” approaches discussed in Section II, which first use outlier detection techniques to identify anomalous tuples in the data set and then use these as supervision signal for a supervised technique. Note that for this approach to be successful it needs to reliably identify a significant fraction of the values that make up the anomaly first, otherwise the subsequent detection of correlated predicates is likely to fail. Therefore, we tested how well different outlier detection mechanisms perform at identifying the data points that correspond to different performance regressions seen in the Windows Azure. We use two real-life incidents with known causes here, which means that for these cases we have exact knowledge of which points are part of the anomaly. The first data set corresponds to anomalies based on slow storage endpoints (which we will describe in detail in the case study in Section VI), and the second data set corresponds to a spike in connection failures due to a software issue.

For both datasets simple outlier-detection techniques that use a constant threshold (such as e.g., a 3σ distance from the average) perform very poorly when labeling anomalies, as they are insensitive to the local ‘context’ (such as changes in the expected latency at different times), yielding, depending on how these thresholds were set, either nearly no tuples labeled as anomalous (when using a 3σ threshold) or, for lower thresholds, mostly generating false positives.

As a consequence, we instead used the well-known LOF technique [23], which takes local context into account, and identifies outliers based on local density measured. This technique uses only a single tuning-parameter – the $MinPts$ parameter governing the size of the local neighborhood – which we varied between 0.05% and 2% of the data set size. We also slightly varied the cutoff-threshold used on the *outlier factor* output by the algorithm (between 1.1 and 1.3).

For the first data set, we found that LOF was able to identify a large faction of the anomalous tuples (depending on the parameter settings, between 77-93%), yet identified a much larger fraction of the non-anomalous tuples as outliers (between 37-40% of the *entire* data set). For the 2nd data set, only 7-53% of the anomalous tuples were correctly labeled, whereas the false positives remained similarly high. As a consequence, we cannot expect a two-step approach based on these outliers to function well for this data.

While this experiment only examines a very limited number of data sets and outlier detection approaches, it does illustrate the difficulty of decoupling the identification of outlier values and mining of the correlated predicates. In contrast, PerfAugur performs both steps jointly and was able to identify, in both datasets, both the known anomalies (among the top 5 anomalies found) as well as predicates pointing to the correct root cause.

VI. Case Studies

To complement the experimental evaluation on synthetic data sets, we describe two case studies illustrating the impact of PerfAugur for two investigations into Windows Azure telemetry.

Case Study 1: “VM deployment latency regression”. One of the KPIs for cloud service providers is the latency at which virtual machines are deployed, as it is imperative to ensure a consistent user-experience. Starting on 7/9/13, we observed a very significant latency regression in one of the clusters, with latencies at the 70th percentile increasing to ∼34 minutes. When this regression was first observed, PerfAugur was not yet available; thus, the initial analysis involved manual analysis of service logs in an attempt to identify the root cause by testing individual hypotheses.

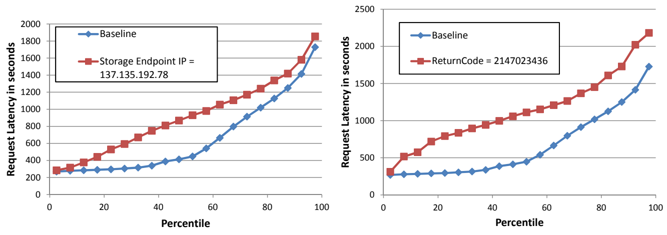


Fig. 10: Top PerfAugur Output for Case 1, identifying storage node IP and return code as indicative of the performance regression.

These hypotheses centered on the suspected root cause being issues in the network stack (e.g., an update to the data center router software, an update to the routing rules and issues with cross cluster network latency), but did not find the cause. Subsequently, a second cluster exhibited a similar regression. At this point, engineers from several independent teams were pulled in and began investigating, with focus on recent node OS updates, background node servicing and the storage stack; again, these investigations did not yield the root cause. The regression was then observed in a 3rd cluster.

Finally, after spending several engineer-weeks of investigation across many teams to isolate the issue, engineers found that in the impacted clusters, a subset of the storage endpoints were experiencing poor throughput. This caused I/O requests performed as part of the VM deployment to have significant delay, and thus delayed the overall VM deployments. Subsequently, the nodes backing these storage endpoints were isolated and a further investigation revealed a BIOS update that resulted in very low server fan speeds in certain situations, causing insufficient cooling and ultimately high temperatures in the blades. This high temperature led to the CPUs throttling clock speed to reduce heat output. As a result, CPU utilization on such nodes could not reach the target of 100%, which resulted in the observed delays.

Once the set of problematic storage endpoints was isolated, the diagnosis of the underlying root case was rapid. However, finding these endpoints required significant time and effort, in part because the fault was not directly tied to a code check-in, in part because the fault only surfaced with a fan configuration unique to that data center, and most importantly because the developers had never seen this regression before.

We used PerfAugur to analyze the same telemetry data, requiring less than 10 minutes for processing. PerfAugur directly identified the individual storage endpoints, with predicates of the form “*StorageEndPointIP = A.X.Y.Z*”, in the top anomalies. In addition, PerfAugur mined a secondary cause of the regression: OS pre-fetch failures (predicate: “*Return_code*

= *-2147023436*") due to timeouts at the storage endpoints, which had been missed before. Starting from this output, identifying the BIOS update as the root cause would have been relatively easy, saving weeks of investigation time.

Case 2: "Understanding the long tail". For our cloud service, the "tail" performance (i.e., high percentiles) of VM deployment operations is crucial, as customers are sensitive to the consistency of the observed performance [24]. In summer 2013, we observed a large differential between the latency at the median ($\sim$5 minutes) and the "long tail" (at the 99th percentile, $\sim$20 minutes). However, the analysis of the deployments "in the tail" is made difficult by the fact that the corresponding instances are made up of many different (but individually rare) *systemic anomalies* as well as independent *transient anomalies*. Historically, developers have chosen issues to fix based on anecdotal evidence, often leading to little or no movement in the tail.

By setting the (categorical) performance indicator to be the (distribution of) "*dominant*" sub-operations (i.e., among all operations involved in a deployment, the sub-operation taking the most time), we were able to automatically identify partitions for which this distribution changed the most over the baseline. We were now able to identify (a) the change in distribution of dominant sub-operations in "tail" deployments and (b) which other conditions correlated with specific sub-operations becoming the bottleneck. One outcome of this analysis was that we discovered the gap between the creation of a VM and the start of its execution to be the dominant sub-operation for a large partition of the data, which in turn lead to the discovery of a new bug in the deployment code.

Summary: Clouds services have KPIs that they need to optimize. These optimizations start with an initial diagnosis, followed by code improvements. Currently, the initial diagnosis phase is often comprised of trial-and-error with simple analytic models, yet still requires multiple engineer-weeks of effort. With PerfAugur, this diagnosis is often completed in minutes, requires no custom development or repeated justification of methodology; moreover, PerfAugur helps overcome human limitations in both the space of hypotheses that is explored as well as the time that is required.

PerfAugur in production: PerfAugur is currently actively used on production data in order to detect performance regressions (using the previous week's data as baseline), detect abnormal increases in error codes (which in turn has allowed us to detect issues with the production system pro-actively) and to support root-cause analysis.

VII. Conclusion

In this paper, we presented PerfAugur – a system for automatically detecting anomalous system behavior and generating explanations in the form of system conditions which co-occur with anomalous operations. While our system was specifically targeted towards service diagnostics in cloud platforms, the algorithms presented here are relevant for any scenario where the underlying data skew and noise requires the use of robust statistics.

VIII. Acknowledgements

We would like to thank Andre Vachon, Mayur Upalekar, Mark Hennessy, Adeel Siddiqui, Henrique Polido, Joseph Chan, Sandi Ganguly, Terence Chan, Amit Kumar, Srivatsan Parthasarathy, Luhui Hu, Royi Ronen, Elad Ziklik and the anonymous ICDE reviewers for their insightful comments on the paper and feedback on the PerfAugur prototype.

References

[1] S. P. Kavulya, S. Daniels, K. Joshi, M. Hiltunen, R. Gandhi, and P. Narasimhan, "Draco: Statistical diagnosis of chronic problems in large distributed systems," *DSN 2013*, vol. 0, 2012.

[2] M. Gabel, A. Schuster, R.-G. Bachrach, and N. Bjorner, "Latent fault detection in large scale services," in *DSN*, 2012.

[3] J. Dean and L. A. Barroso, "The Tail at Scale," *Commun. ACM*, vol. 56, no. 2, Feb. 2013.

[4] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, "Bobtail: Avoiding long Tails in the Cloud," in *NSDI*, 2013.

[5] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: trading a little bandwidth for ultra-low latency in the data center," in *NSDI*, 2012.

[6] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "Detail: reducing the flow completion time tail in datacenter networks," in *SIGCOMM*, 2012.

[7] K. Nagaraj, C. Killian, and J. Neville, "Structured comparative analysis of systems logs to diagnose performance problems," in *NSDI'12*, 2012.

[8] C. C. Aggarwal, *Outlier Analysis*. Springer, 2013.

[9] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection*. Wiley, 1987.

[10] E. Wu and S. Madden, "Scorpion: Explaining away outliers in aggregate queries," *Proc. VLDB Endow.*, vol. 6, no. 8, Jun. 2013.

[11] A. X. Zheng, J. Lloyd, and E. Brewer, "Failure diagnosis using decision trees," ser. ICAC '04, 2004.

[12] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase, "Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control," in *OSDI*, 2004.

[13] S. Duan, S. Babu, and K. Munagala, "FA: A System for Automating Failure Diagnosis," in *IEEE ICDE*, 2009.

[14] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: Automated classification of performance crises," ser. EuroSys '10, 2010.

[15] S. Roy and D. Suciu, "A formal approach to finding explanations for database queries," ser. SIGMOD, 2014, pp. 1579–1590.

[16] B. Micenková, R. T. Ng, X. H. Dang, and I. Assent, "Explaining outliers by subspace separability," in *ICDM*, 2013.

[17] E. Müller, M. Schiffer, and T. Seidl, "Statistical selection of relevant subspace projections for outlier ranking," in *ICDE*, 2011, pp. 434–445.

[18] S. M. Qiang Ma and M. Sandler, "Frugal streaming for estimating quantiles," *Space-Efficient Data Structures, Streams, and Algorithms*, 2013.

[19] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "Approximate Medians and other Quantiles in one Pass and with limited Memory," in *ACM SIGMOD*, 1998.

[20] M. D. Atkinson, J.-R. Sack, N. Santoro, and T. Strothotte, "Min-max Heaps and Generalized Priority Queues," *CACM*, vol. 29, no. 10, Oct. 1986.

[21] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951.

[22] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics.*, 2000.

[23] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, no. 2, 2000.

[24] M. Mao and M. Humphrey, "A performance Study on the VM Startup Time in the Cloud," in *IEEE CLOUD*, 2012, pp. 423–430.