Anode: Empirical Detection of Performance Problems in Storage Systems using Time-Series Analysis of Periodic Measurements

Vipul Mathur
NetApp India Private Limited
vipul.mathur@netapp.com

Cijo George
NetApp India Private Limited
cijo.george@netapp.com

Jayanta Basak
NetApp India Private Limited
jayanta.basak@netapp.com

Abstract—**Performance problems are particularly hard to detect and diagnose in most computer systems, since there is no clear failure apart from the system being slow. In this paper, we present an empirical, data-driven methodology for detecting performance problems in data storage systems, and aiding in quick diagnosis once a problem is detected.**

The key feature of our solution is that it uses a combination of time-series analysis, domain knowledge and expert inputs to improve the overall efficacy. Our solution learns from a system's own history to establish the baseline of normal behavior. Hence it is not necessary to determine any static trigger-levels for metrics to raise alerts. Static triggers are ineffective since each system and its workloads are different from others.

The method presented here (a) gives accurate indications of the time period when something goes wrong in a system, and (b) helps pin-point the most affected parts of the system to aid in diagnosis. Validation on more than 400 actual field support cases shows about 85% true positive rate with less than 10% false positive rate in identifying time periods of performance impact before or during the time a case was open. Results in a controlled lab environment are even better.

I. Introduction

Performance problem diagnosis in storage systems is an expert centric process today. The basic steps start with gathering metrics and reports about the system in question. Then a human expert sifts through the data to find various levels of evidence of a possible cause of the reported problem. Once the problem has been diagnosed, the expert recommends a course of action to work-around or resolve the issue. The success of this process depends heavily on the availability and abilities of the human experts. In industrial settings, often various levels of such manual assessments are done to triage the common recurring issues quickly at the first level itself, and escalate the tricky ones to more proficient/ specialized performance experts. Thus there is a large spread in the time it takes to resolve a problem reported by the customer (case age). This is illustrated in Figure 1 by the standard box and whisker plots of case age from about 100,000 reported incidents. Note that the median (thick line in the figure) is $10\times$ higher for performance related cases compared to the median for other cases.

As the complexity of storage solutions grows in order to provide better features and functional capabilities, it becomes paramount to scale the diagnosis capabilities as well. With the advent of better data gathering, processing and analysis capabilities, there is an opportunity to (a) automate repetitive tasks involved in diagnosis; (b) catch common well-known problems early and automatically; and (c) aid experts in study and diagnosis of the harder/ less-common problems. Reducing the burden on human experts, from various parts of the detection and diagnosis process, would provide a means to scale overall support and problem resolution. One way to reduce the turn-around time for resolving performance problems is to notice the problems as early as possible and then narrow-down the affected parts of the storage system.

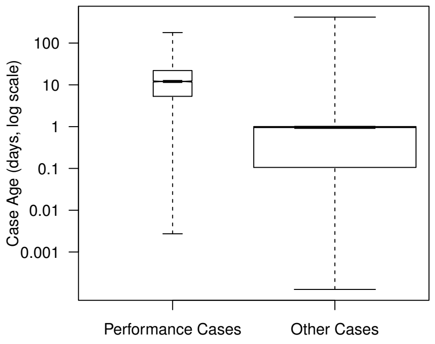


Fig. 1. Performance cases stay open $10\times$ longer (median) than other cases.

There are two key challenges that need to be addressed. First, there is often a gap between the start of a problem and its impact being noticed by users. For instance, see Figure 2 that depicts two metrics from a single system over multiple weeks. Note that the time when the incident was reported (case filed) is much later than the first time when a deviation in regular pattern (anomaly) is detected. Second, static thresholds for raising alerts are insufficient. A threshold being crossed at peak load may be *normal*, and not indicative of a problem (e.g. weekly periodic peaks in both the metrics in Figure 2). Also, instead of a value above the threshold indicating a problem, a missing high peak in the metric may be the observed effect of an underlying problem. This is seen in the Disk Reads metric (see Figure 2) after the time that the case was filed. These and other challenges are addressed by our data-driven method to enable early detection of performance problems (ideally before the impact is felt/ noticed by the users) and also ways to

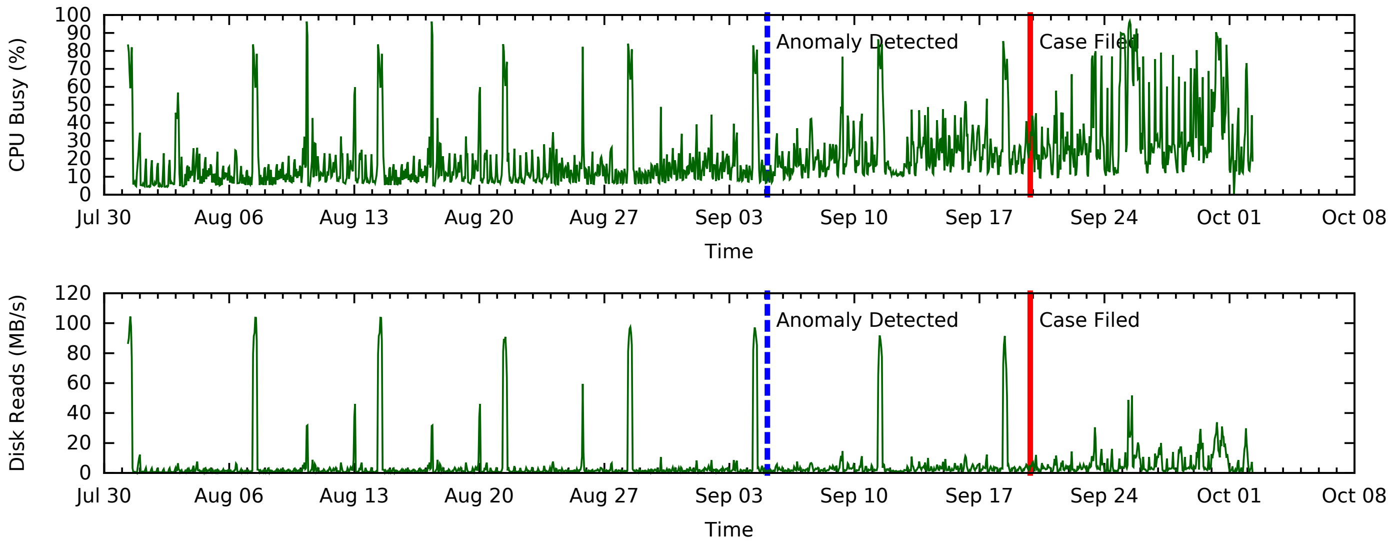


Fig. 2. Nine weeks of CPU Busy and Disk Reads metrics from a field-deployed storage system. *Anode* detects an anomaly in the system much before the user actually opened a support case, as indicated.

identify the top affected parts of the system. The key highlights and contributions of this work are:

- Using the history of a system to derive a dynamic notion of normal behavior. Hence, eliminating the need to set static thresholds on metrics for raising performance alerts.
- Applying an empirical, data-driven approach to detection of performance problems in storage systems hosting actual customer workloads.
- The core of our method is metric, workload and system agnostic and can be readily adapted to other types of server systems.
- We take domain knowledge into account by using the notion of aggregation sets that can be pre-determined.
- This method aims to make the life of support personnel and administrators easier, rather than aim to replace them with a complicated system.
- The system uses simple, well-known concepts for its core processing and thus can readily keep up with changes in system needs and hardware/ software stack. This is often not the case for other methods with brittle models of system behavior.

The paper is organized as follows. Section II discusses some existing work pertaining to applications and techniques for anomaly detection. Section III gives an overview of the architecture and system model for the detection of performance problems in field-deployed storage systems. We describe our proposed methodology in Section IV, and present results on lab experiments and field-reported incidents in Section V. The paper concludes in Section VI.

II. Related Work

In literature anomaly detection has been used in varying application domains starting from credit card fraud detection to network anomaly detection. Below we first discuss some specific use-cases of anomaly detection, followed by a summary of techniques for anomaly detection on time-series data.

A. *Applications of anomaly detection*

In [1], network anomalies are detected using a modified form of kernel recursive least square. The authors first detect the normal range of values in the feature space (upper and lower bounds). Subsequently, any observation violating these bounds is triggered as an anomaly. In [2], a probabilistic model of the time-series of observations is constructed using exponentially weighted moving average (EWMA). If any observation deviates from the value predicted by this model, then an anomaly is flagged. This technique is then successfully applied in network anomaly detection. In [3], multivariate singular value decomposition (SVD) technique has been used to represent the time-series of network traffic into multiple principal components. Any deviation in the normal ranges of these multiple components is detected as a network anomaly.

In [4] anomalies in the Internet traffic have been identified by representing the traffic as multi-resolution time-series. Any anomaly detected at any resolution is then summed up (with the proper weighting mechanism) with that of the other resolutions. The authors claim that the resultant anomaly detection is much superior to single-resolution anomaly detection. Anomaly detection has been successfully applied in various other domains such as automobile fault detection [5], aviation safety [6], early detection of ecosystem disturbances in earth science data [7], and early detection of disease outbreak [8].

Anomaly detection has been successfully applied in problem diagnosis in different systems. In many of these analyses, certain signatures were computed from the problem sources (anomalies) and then these signatures were visually displayed or automatically processed. For example, in [9], the authors developed a visualization tool called Theia that analyzes application-level logs in a Hadoop cluster, and generates visual signatures of each job's performance. These visual signatures provide compact representations of task durations, task status, and data consumption by jobs. In [10], the authors developed a diagnostic tool called Giza to characterize and troubleshoot performance issues in one of the largest IPTV networks in North America. Various measurements such as device usage and error logs, user activity logs, video quality alarms, and customer trouble tickets were considered and multi-resolution data analysis was applied to detect and localize regions in the IPTV distribution hierarchy that are experiencing serious performance problems. In [11], the authors use machine learning, specifically a decision tree based random forests, to characterize applications' hardware behavior by modeling the hardware events. They use machine learning on the architectural micro-benchmarks to fingerprint the performance problems throughout the core and memory hierarchy.

To our knowledge, this paper represents the first application of time-series analysis based anomaly detection combined with domain knowledge to analyze empirical data for detecting performance problems in storage domain.

B. *Techniques for anomaly detection in time-series*

The techniques for anomaly detection in time-series have been broadly classified into the following four categories as per survey presented in [12]–[14]: (a) kernel based techniques, (b) predictive techniques, (c) segmentation based techniques, and (d) window based techniques. Usually in kernel based techniques, the time series is modeled as a function of kernel matrices of the observations in multidimensional space. The test time series is then fitted with the function using optimization techniques and outliers are obtained as the anomalous vectors of observations. For example, support vector regression is an instance where some of the support vectors can represent the anomalous behavior. In predictive techniques, the time-series is statistically represented as a function of its past behavior (for example EWMA in [2]). The future values are then predicted based on the past behavior and any major deviation from the predicted behavior is considered as an anomaly. In segmentation based techniques, a time-series is represented as short segments and a model is built to represent the transitions as state-transition machine (e.g., Markov chain). Any behavior manifested as a transition that cannot be captured by the state-transition machine is treated as an anomaly. In the window based techniques, several windows of observations are overlaid on each other and the normal behavior with respective bounds are derived. Any characteristic of the time-series not conforming the bounds of normal behavior is flagged as anomaly. In this paper, we perform widow-based anomaly detection.

III. Architecture and System Model

The parts involved in detection and diagnosis of performance problems in deployed storage systems are depicted in Figure 3. Metrics are periodically (say every hour) collected and recorded locally by the storage systems deployed at the field data center. This recorded measurement data is uploaded to the central *Anode* data center via the Internet, and sits in a database. Next, the data is pulled out of the database and analyzed by the *Anode* system in a batch manner. The specific steps in processing are depicted as blocks in Figure 3 and described later in Section IV. The results of the analysis are in the form of alerts or advice that can be sent back to administrators at the field deployment site, or alternatively consumed by a reporting and corrective action system.

The *Anode* system consists of a data warehouse to store the measurement data uploaded from all the systems deployed

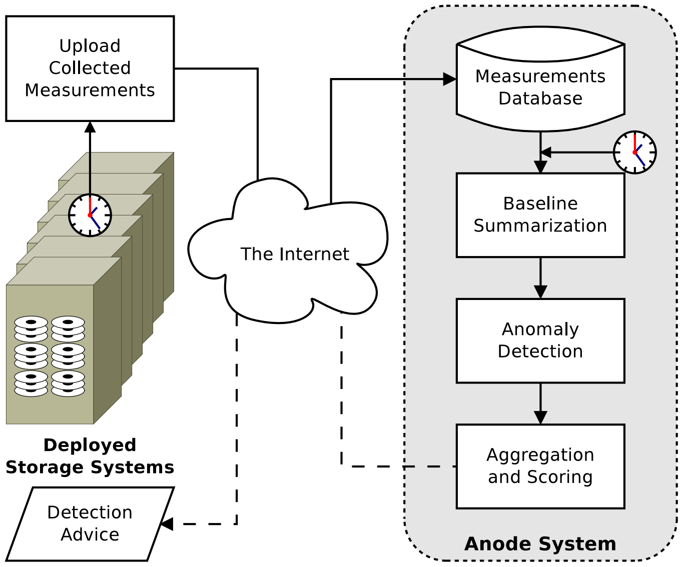


Fig. 3. Overview of performance problem diagnosis, in deployed storage systems, using *Anode*.

in the field data centers. It also contains a cluster of compute nodes where the data is processed. In our current experimental setup, the compute nodes are represented by a single virtual machine with enough resources to completely process one week's measurement data from one storage system in about 5-10 minutes. More such virtual machines can be provisioned, as the need arises, to analyze more systems in parallel.

The current work-flow for addressing any performance issues that may arise in a deployed storage system has been briefly introduced in Section I. With the *Anode* system, the main change in this process would be that data is automatically analyzed at a central location (instead of local tools at the field data center), and the results are made available to the admins.

Looking deeper into the storage systems themselves, there are various sub-systems/ resources/ logical entities that need to be monitored. Hence, each sub-system is programmed to monitor metrics relevant to that sub-system. The full set of metrics is thus divided into sub-sets based on hardware resources (e.g. disks, processors, network cards, memory etc.), or logical/ software entities (such as RAID groups, partitions, protocols etc.). Further, most of these entities (for example: processors, disks, storage space partitions) have multiple instances within each system. The same sub-set of metrics specific to each sub-system (like utilization for CPU, throughput for network card, latency for protocols, space used for partitions) is monitored for each instance of that entity. In the next section, we describe the steps of our solution in detail.

IV. Solution Methodology

Our solution consists of the steps and parts depicted in Figure 3. Before describing the steps involved in our solution below, we begin with a brief introduction to the metrics that are measured. We also introduce some common facts and notations used in the rest of the paper.

In our study, the number of total metrics per storage system varied from a few hundred for smaller systems to several tens of thousands for large systems with thousands of disks and multiple network cards etc.

Since metrics are collected periodically over time (hourly averages in this case), it makes sense to view each metric as a time series of individual measurements. Let us introduce some notations used to describe the analysis in the rest of the paper. We denote a time series of measurements of metric x (taken at regular discrete intervals Δt) with bold lowercase letters like $\boldsymbol{x}$. We have

$$\boldsymbol{x}^{\boldsymbol{\Delta t}} = (x_1, x_2, \ldots, x_n) \tag{1}$$

where x_i for $1 \le i \le n$ are n values of the metric x. The single discrete measurement x_i of metric x taken at time instant t represents an average of the values of the quantity x in the previous left-open interval $(t - \Delta t, t]$. Thus if measurement x_1 was recorded at time t, then x_2 would be recorded at time $t + \Delta t$, x_3 at time $t + 2\Delta t$ and so on till x_n being recorded at time $t + (n-1)\Delta t$. Since the time interval between measurements is always one hour throughout this paper, we drop the Δt notation and call the time series $\boldsymbol{x}$ for simplicity. A set of multiple related time series is denoted by bold uppercase letters like $\boldsymbol{X}$, and defined as

$$\boldsymbol{X} = \{\boldsymbol{x}, \boldsymbol{y}, \ldots\}. \tag{2}$$

A. Data Collection

In order to analyze the value of metrics over time, we use a two-step scheme for collecting the measurement data itself. First, each storage system periodically measures metrics of interest (every hour in this paper, but any appropriate fixed period could be used) and internally records them to a log file. Second, these recorded measurements are sent back to the *Anode* data center via the Internet over a secure channel. This can be done either at regular intervals (say daily/ weekly), or on-demand by an administrator at the field data center. The *Anode* data center receives these collected measurements from all deployed storage systems across the world and stores them in a data warehouse, ready for analysis. Note that the set of metrics to monitor and record is programmed into each storage system before it is deployed. Hence there are no direct calls made from the *Anode* data center to the actual storage systems in the individual field data centers.

In the following subsections we describe the analysis steps that are carried out by *Anode*. These steps are carried out in batch mode on the collected data read from the data warehouse (see Figure 3). The analysis may also be manually triggered if needed, for instance, when a field administrator calls for an assessment.

B. Baseline Summarization

The objective of this step is to establish the expected range of values for each metric. This is where we avoid the use of any static thresholds, since they are observed to be insufficient as discussed earlier.

a) Weekly periodicity: of metric values is observed in most systems we studied. This means that the same pattern of metric values is seen to repeat week after week. Thus, for instance, the average value each week for the period Monday 9am-10am is expected to be in the same range as previous week's Monday 9am-10am. A sample of this is seen visually in Figure 4 where the Rate of Read Operations is seen to repeat in the same pattern week after week. This weekly periodicity is the key observation we utilize to find the expected baseline *normal* value for each metric, at each hour of a week. Note that the underlying reason for such periodicity is that workloads are triggered by users' actions, that are known to go through daily cycles, with weekends being different.

Our summary $\mathcal{S}(\boldsymbol{x})$ of what a typical week looks like for metric x represented as time series $\boldsymbol{x}$ is based on the previous k weeks of values for the metric, summarized as follows. If $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^k$ are the time-series of values of metric x in previous k weeks, where

$$\boldsymbol{x}^k = \left(x_1^k, x_2^k, \ldots, x_n^k\right) \tag{3}$$

and

$$\widetilde{x_i} = \text{median}\left(x_i^1, x_i^2, \ldots, x_i^k\right) \tag{4}$$
$$\sigma_{x_i} = \text{standard\_deviation}\left(x_i^1, x_i^2, \ldots, x_i^k\right) \tag{5}$$
$$\widetilde{\boldsymbol{x}} = (\widetilde{x_1}, \widetilde{x_2}, \ldots, \widetilde{x_n}) \tag{6}$$
$$\sigma_{\boldsymbol{x}} = (\sigma_{x_1}, \sigma_{x_2}, \ldots, \sigma_{x_n}) \tag{7}$$

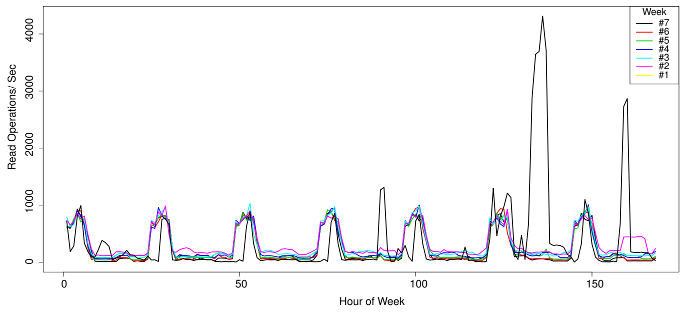
then

$$\widetilde{\boldsymbol{x}} \pm \sigma_{\boldsymbol{x}} = (\widetilde{x_1} \pm \sigma_{x_1}, \widetilde{x_2} \pm \sigma_{x_2}, \ldots, \widetilde{x_n} \pm \sigma_{x_n}) \tag{8}$$
$$\mathcal{S}(\boldsymbol{x}) = \{\widetilde{\boldsymbol{x}},\ \widetilde{\boldsymbol{x}} - \sigma_{\boldsymbol{x}},\ \widetilde{\boldsymbol{x}} + \sigma_{\boldsymbol{x}}\} \tag{9}$$

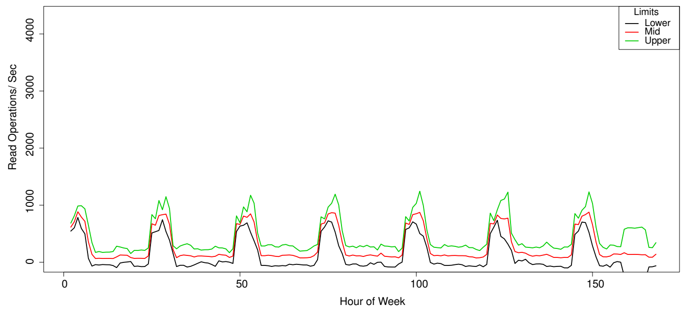
In other words, the baseline summary of a metric consists of the median and a 2σ band at each index period (hour) of the week, based on data from the same period in previous k weeks. Note that since we have hourly values as our measurements, there are n=168 successive values in each week. As sample of such a summary, see the first two graphs of Figure 4 depicting the Rate of Read Operations in a system. The first graph (Fig. 4(a)) depicts successive weeks with overlapping lines illustrating the periodic nature of the workload, and the second graph (Fig. 4(b)) depicts the median, upper and lower bands of the summary of the same metric.

We chose median as our measure of central tendency as it is robust to outliers, while having standard deviation (not robust to outliers) as a measure of dispersion. This allows for a liberal estimate of the acceptable range of metric values. Another option is to use inter-quartile range as the measure of dispersion, and this would make the range more conservative in the case of large deviations (outliers). Since the k weeks of data may naturally have some non-periodic behavior, it would help to filter out slow trends and irregular behavior using a time-series decomposition approach—retaining only the seasonal (periodic) portions of the metric. This option is available in our method, however is optional for use since we may choose the reference k weeks from a time when the system was known to operate within acceptable limits.

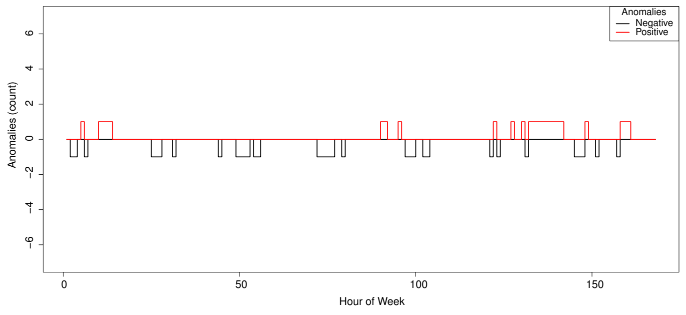
Apart from the choice of using decomposition-based filtering or quartile based dispersion measures, one needs to specify the amount of history to process (k weeks) and where to take the historical data from. In our evaluations on real customer field data, k=4 weeks of data was found to be enough. Also, *Anode* allows for storing the summary time-series $\mathcal{S}(\cdot)$ and retrieving it later for analysis. This allows for having a



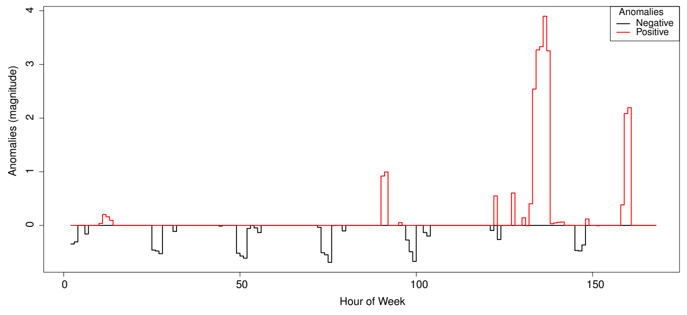
(a) Periodic metric behavior (week over week)



(b) Weekly summary (based on weeks #1-#4)



(c) Marked anomalous hours (flags) in week #7



(d) Marked anomalous hours (magnitudes) in week #7

Fig. 4. Illustration of baseline summarization and anomaly detection steps for system-wide read operations per second from an actual field system.

known good summary being used repeatedly for assessment, and revised as needed.

C. Anomaly Detection

The next step is to use the summary $\mathcal{S}(\cdot)$ to mark deviations from this expected behavior in the duration being tested. Any value which lies outside the band of expected values is called an anomaly. We represent such anomalies as the value of an indicator function $\mathcal{A}(\cdot)$ defined as:

$$\mathcal{A}(x_i) = \begin{cases} 1 & \text{if } x_i > \widetilde{x}_i + \sigma_{x_i} \\ 0 & \text{if } \widetilde{x}_i - \sigma_{x_i} \leq x_i \leq \widetilde{x}_i + \sigma_{x_i} \\ -1 & \text{if } x_i < \widetilde{x}_i - \sigma_{x_i} \end{cases} \tag{10}$$

thus a time series $\boldsymbol{x}$ of metric values has a corresponding time series of anomaly indications represented as:

$$\mathcal{A}(\boldsymbol{x}) = (\mathcal{A}(x_1), \mathcal{A}(x_2), \ldots, \mathcal{A}(x_n)) \tag{11}$$

Naturally, anomalies may be positive (if the actual value is above the upper band from $\mathcal{S}(\cdot)$ for that hour of the week) or negative (if value is below the lower band). This can be seen in Figure 4(c) that shows the anomaly time series $\mathcal{A}(\cdot)$ for our sample metric. Apart from the existence of anomalous values indicated by $\mathcal{A}(\cdot)$, we are also interested in a measure of the severity of the anomalous values. For this, we define a *normalized anomaly magnitude* function $\mathcal{M}(\cdot)$ that measures the amount of deviation of a metric from the band of acceptable values $\mathcal{S}(\cdot)$ at each time index i as:

$$\mathcal{M}(x_i) = \frac{x_i - (\widetilde{x}_i + \sigma_{x_i}\,\mathcal{A}(x_i))}{\max\,(x_i^1, x_i^2, \ldots, x_i^k)} \tag{12}$$

In this manner, we are essentially looking at the amount by which the metric value x_i falls outside the upper/ lower band of acceptable values (see Eq. (9)). In order for magnitudes to be comparable across metrics, we choose to normalize this deviation to the maximum value observed for the metric in the weeks used to create the baseline summary $\mathcal{S}(\cdot)$. This gives rise to an anomaly magnitude time series represented as:

$$\mathcal{M}(\boldsymbol{x}) = (\mathcal{M}(x_1), \mathcal{M}(x_2), \ldots, \mathcal{M}(x_n)) \tag{13}$$

This is a time series of positive and negative anomaly magnitudes as depicted in Figure 4(d).

D. Aggregation and Scoring

The previous steps dealt with time series of measured values of each individual metric, and marked anomalies in those values during the duration being assessed. We used a metric-specific summary created from past values of each metric. However, individual metrics indicating an anomalous value may be noisy/ error-prone and may or may not constitute a problem worth being escalated or considered significant. In order to suppress sporadic fluctuations in individual metrics that don't indicate a bigger problem, we make use of the fact that metrics in the storage system being monitored are naturally grouped in many ways. The strategy we use is to combine the anomaly time series $\mathcal{A}(\cdot)$ and normalized anomaly magnitudes $\mathcal{M}(\cdot)$ of *multiple related metrics* appropriately (as described below). This serves the dual objectives of (a) creating more robust indications of when a problem occurs, and also (b) creating a way to assess specific sub-systems (storage partitions, disks, processors, protocols) within the given storage system.

1) Aggregation: First, we define an aggregation set $\mathcal{G}$ as a set of related metrics chosen from domain knowledge of the storage system internals. The set $\mathcal{G}$ is essentially a set of individual metric time series

$$\mathcal{G} = \{\boldsymbol{x}, \boldsymbol{y}, \ldots\} \tag{14}$$

and $\#\mathcal{G}$ is the cardinality of the set, i.e. the number of metrics in the aggregation set. These metrics may be of the same type (say latency at various points in the system), or correspond to the same sub-system (say all metrics corresponding to a particular disk or partition). This grouping may also be chosen from the knowledge of which metrics most effectively indicate a particular type of problem.

Next, for each specified aggregation set $\mathcal{G}$, two time series are derived as follows. First, we define an anomaly count time series $\mathcal{C}(\cdot)$ that counts the number of metrics out of the set $\mathcal{G}$ that have an anomaly at each time index i (where $1 \leq i \leq n$, and n is the length of the assessment period) as:

$$c_i = \sum_{g \in \mathcal{G}} |\mathcal{A}(g_i)| \tag{15}$$

$$\mathcal{C}(\mathcal{G}) = (c_1, c_2, \ldots c_n) \tag{16}$$

Note that we have used the absolute value $|\mathcal{A}(\cdot)|$ of the anomaly indicator function, since we don't differentiate a negative anomaly from a positive anomaly for the aggregation step.

Similarly, we also define an aggregated anomaly magnitude time series $\mathcal{L}(\cdot)$ by averaging the normalized anomaly magnitudes $\mathcal{M}(\cdot)$ of the metrics in $\mathcal{G}$. We have, for $1 \leq i \leq n$,

$$l_i = \sum_{g \in \mathcal{G}} \frac{|\mathcal{M}(g_i)|}{\#\mathcal{G}} \tag{17}$$

$$\mathcal{L}(\mathcal{G}) = (l_1, l_2, \ldots l_n) \tag{18}$$

Note that such a combination makes sense since each individual anomaly magnitude $\mathcal{M}(\cdot)$ is already normalized.

At this stage, we further employ a threshold based scheme for assessing significant anomalies within a given aggregation set $\mathcal{G}$. This is done via a significance indicator function $\mathcal{F}(\cdot, \cdot)$ based on $\mathcal{L}(\cdot)$ defined above. For this we externally configure two thresholds π and θ where $P_\pi(\cdot)$ denotes the π^{th} percentile value of the given set of numbers, and $\theta \geq 1$ is a threshold on the average anomaly magnitude within the set $\mathcal{G}$. Then we have:

$$\mathcal{F}(\mathcal{G}, i) = \begin{cases} 1 & \text{if } (l_i \geq P_\pi(\mathcal{L}(\mathcal{G})) \text{ and } (l_i \geq \theta) \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

This gives us the final time series $\boldsymbol{F}(\mathcal{G})$ of raised anomaly flags for the aggregation set $\mathcal{G}$ as:

$$\boldsymbol{F}(\mathcal{G}) = (\mathcal{F}(\mathcal{G}, 1), \mathcal{F}(\mathcal{G}, 2), \ldots, \mathcal{F}(\mathcal{G}, n)) \tag{20}$$

Appropriate choice of the parameters π and θ by users can make *Anode* liberal or conservative in assessing whether an anomaly is significant or not. Typical values are $\pi = 50$, $\theta = 2$ for a liberal assessment and $\pi = 80$, $\theta = 4$ for more conservative setting.

2) Scoring: There can be many possible such aggregation sets of metrics in a system. For instance, we may want to create an aggregation set for each storage space partition in the system. Then the count $\mathcal{C}(\cdot)$, combined anomaly magnitude $\mathcal{L}(\cdot)$ and raised anomaly flags $\boldsymbol{F}(\cdot)$ will be known, as defined above, for each of the aggregation sets. The next step in our methodology is to come up with a score for each set, so that different aggregation sets may be compared with each other on some factors. Each score is a single positive scalar number calculated over the assessment period (n observations). There are several ways to come up with such a score, and we describe a few that are implemented in *Anode*.

a) Total Anomaly Duration: (TAD) is the simplest score, and counts the number of time periods in which an anomaly is flagged in the given aggregation set. This is expressed as:

$$\text{TAD}(\mathcal{G}) = \sum_{i=1}^{n} \mathcal{F}(\mathcal{G}, i) \tag{21}$$

b) Cumulative Anomaly Magnitude: (CAM) is the sum of anomaly magnitudes $\mathcal{L}(\cdot)$ in all the time indices of the assessment period, for the given aggregation set. This is expressed as:

$$\text{CAM}(\mathcal{G}) = \sum_{i=1}^{n} \sum_{g \in \mathcal{G}} \frac{|\mathcal{M}(g_i)|}{\#\mathcal{G}} \tag{22}$$

c) Mean Anomaly Count: (MAC) is another simple measure that counts total number of metrics that raise an anomaly in each index of the assessment period, and then divides by the number of metrics involved in the set. Expression for MAC is as follows:

$$\text{MAC}(\mathcal{G}) = \sum_{i=1}^{n} \sum_{g \in \mathcal{G}} \frac{|\mathcal{A}(g_i)|}{\#\mathcal{G}} \tag{23}$$

E. Types of Analysis Output

The core *Anode* methodology described here can be used for carrying out several kinds of analysis. A single system can be analyzed (on-demand, or automatically) for a given assessment duration to determine if a performance problem exists in the system, and if so where. The key enabling factor in each of the following analysis is the use of anomalies detected using the *Anode* method described earlier.

1) Flag time-periods of impact: This is the basic output meant to identify time periods within the requested assessment duration, when a performance problem has been flagged. Thus, for instance, an intermittent performance impact might be found for 3 of the 200 hours that were analyzed. Each subsystem/ resource in the system can be assessed in this manner, by using the corresponding set of metrics as an *aggregation set* (see Section IV-D).

This basic flagging of impacted time-periods allows further analysis to be triggered, and establishes a time-line of performance impact. This is beneficial also in the case that an expert needs to later look at the history of the problem, much after the actual impact occurs.

2) Pin-pointing affected parts: Once some flag has been raised for any of the analyzed entities within the system, the corresponding score for each entity (see Section IV-D2) can be used to get an ordered short-list of affected entities that were flagged anywhere in the requested assessment duration. Thus, *Anode* helps not only in pin-pointing what parts of the system are affected (i.e. having a non-zero score), but also identifies an *ordered* list of most affected parts. For instance, one performance problem may impact several storage partitions hosting data. By looking at the scores for these partitions in descending order, the most affected partitions can be pointed out.

This is often an important step in identifying what is the impact of a problem on the availability of stored data. Identifying the most affected resources, data partitions, protocols etc. will often lead to quick diagnosis by focusing the attention of an administrator/ support expert to the correct place within a large system.

3) Identify top symptoms: A useful ability of *Anode* is the identification of top *symptoms* i.e. indicators of a performance problem experienced by a system. Indicators of hardware failures are usually clear—for instance a failed disk—but the same may not be true for performance problems. The root cause of the problem may be hidden and only its impact on different subsystems is seen.

Anode leverages its scoring mechanism (see Section IV-D2) to find the list of most affected metrics by applying it to each metric individually within each system that is analyzed. This list of top affected metrics in the system can often reveal the underlying cause quickly, when seen by an experienced administrator/ support engineer. For instance, if the metric for cache hit rate is showing an anomaly along with an anomalous increase in latency, then it makes sense to look if the nature of the workload changed in a way to make the system's cache less effective.

This section described the core analysis used in *Anode*. In the next section, we present validation, results and discussion for using *Anode* in both a controlled environment, as well as on historical data from actual production deployed storage systems.

V. Experiments and Field-Validation

In the following subsections, we demonstrate the efficacy of the *Anode* methodology by (a) using controlled experiments in a laboratory environment to establish the accuracy of anomaly detection when the cause and time of occurrence of the problem is known, and (b) by using actual past measurements from hundreds of storage systems deployed in production datacenters around the world to evaluate how our method fares in real world conditions, where the cause and time-periods of a problem are not known *a-priori*.

A. Lab validation

The first step in our evaluation of the *Anode* methodology is to apply it in a controlled laboratory environment. This gives us two advantages over the use of field data. First, since the problem is created as part of our experiment, the cause and time-periods when the problem exists in the system is known.

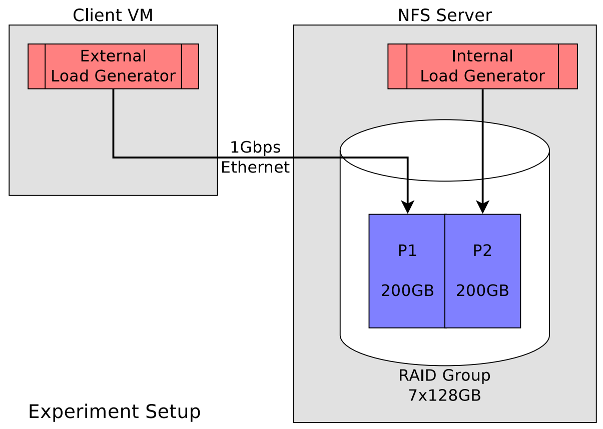


Fig. 5. Setup for laboratory validation

This gives good ground-truth to evaluate the effectiveness of *Anode*. Secondly, a controlled environment helps in tuning some of the parameters of *Anode* to improve the accuracy of our method. Later, we also present results from actual field measurements, where we do not have existing periodic (hourly) assessments of the existence of a problem.

Our experiment setup (see Figure 5) consists of an NFS storage server with 16GB RAM, and a client virtual machine (VM) running Ubuntu 9.10 (Linux kernel version 2.6.31-22). For the experiments, we use a logical partition P in the storage system created by a RAID group consisting of 7 disks of 128 GB each, configured with dual parity RAID to protect against up to two disk failures. Two logical sub-partitions P1 and P2, 200 GB each, are created on partition P. 100 files with random data, 492 MB each, are created on P1. Total data set size for P1 is thus 48 GB, which is three times the storage system's RAM. This is to make sure that caching within the NFS server will not dominate the outcome of our experiments. A single 32 GB file with random data is created on P2. The sub-partition P1 is then exported via NFS and mounted directly within the client VM.

A workload generation tool in the client VM is used to generate I/O traffic on the files in P1. Even though these are essentially micro-benchmarks in a controlled lab environment, we still wanted the I/O patterns to mimic those observed in real setups. Hence, our workload generator (called *Patio*) emulates four concurrency patterns (*A, B, C, D*) observed from actual deployments (for instance, see Figure 6). The specifications of workloads with varying parameter combinations is given in Table I. These are chosen from commonly observed values seen in the field. Thus a combination of one concurrency pattern and one workload gives us a description of the input I/O traffic used in a single experiment.

Since our method detects performance impact, we also need ways to artificially create a disturbance in our experiment setup. For this we follow three scenarios:

- **Secondary workload** scenario uses another load generator (internal to the NFS server, see Figure 5) to generate a disturbance workload on P2 to emulate a performance problem in the storage system caused by interference from any workload apart from the primary client VM. We use two sets of such internal workload called *Anomaly-1* and *Anomaly-2* in Table I.

TABLE I
EXPERIMENT WORKLOAD DESCRIPTION

Workload	Read Ops (%)	Random Seek (%)	Concurrency
Client-1	60	80	*A, B, C, D*
Client-2	60	20	*A, B, C, D*
Anomaly-1	100	80	128
Anomaly-2	0	80	128

- **Degraded RAID** scenario artificially fails one disk in the RAID group, while no spare disks are available. Hence, the RAID subsystem has to reconstruct the data of the missing disk on-the-fly by using the parity information. This is denoted as *Anomaly-3*.
- **RAID reconstruction** scenario is an extension of degraded RAID, where a spare disk is added to the system. Then the RAID system starts reconstructing the data of the disk that had failed earlier, on to the new spare disk. While the system can intelligently do this in the background to avoid impacting the foreground workload, we force immediate reconstruction to make the impact more direct, for the purposes of our test. For simplicity, this is always combined with *Anomaly-3* in the same experiment, to give three overall combinations of disruptions.

In this manner, we used four input concurrency patterns, along with two workload combinations, and also three sets of artificially injected disruptions. This leads to 24 experiments, each of which was repeated four times. Each experiment is structured as follows (see example in Figure 6).

- Weekly periodic customer workload pattern is emulated on the storage system for a period equivalent to five weeks and periodic averaged measurements of metrics from the system are collected. To keep time tractable, an hour is scaled down to a minute. Hence the customer workload pattern repeats every 168 minutes (representing 168 hours of a week) for a period of 168*5=840 minutes (representing five weeks) as shown in the first two graphs of Figure 6.
- A problem is triggered for some period of time once or multiple times during the fifth week of the emulated customer workload and the start and end times of each triggered anomaly is recorded to create a disturbance time-line (for instance, graph three in Figure 6).
- The collected periodic metric measurements from the system are analyzed using *Anode* by taking the first four weeks as the reference period for creating a summary and the last week is assessed for anomalies. The resulting anomaly time-line output is then compared with the recorded disturbance time-line hour-by-hour (minute-by-minute in our experimental setup).

1) Per-experiment results: The following standard statistics are collected for each experiment:

- True Positives (TP): Number of periods where anomaly was indicated by *Anode* matching with the actual anomaly time line.
- False Positives (FP): Number of periods where anomaly was indicated by *Anode*, but there wasn't any actual anomaly present. And similarly,

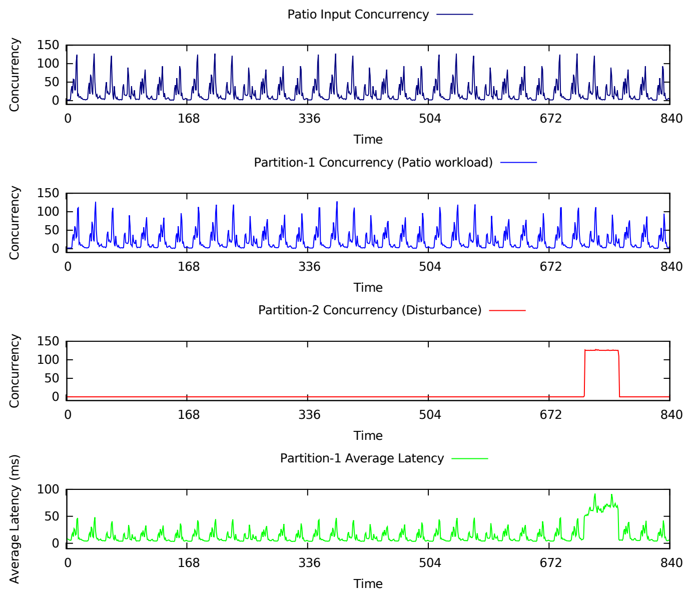


Fig. 6. Time-lines from one laboratory experiment. The disruption created on P2 in the end is clearly seen to impact the latency of P1.

- True Negatives (TN): *Anode* didn't indicate an anomaly and none was actually present.
- False Negatives (FN): *Anode* didn't indicate an anomaly when there was an actual disruption.

These can be used to derive the following set of standard statistical measures, which have their usual meaning from statistics:

$$\text{True Positive Rate (TPR)} = \frac{TP}{TP + FN} \tag{24}$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP + TN} \tag{25}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{26}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN} \tag{27}$$

As mentioned earlier, an assessment (along with the above stats) is available for each *aggregation set* (see Section IV-D). For instance, in our experiment setup, we generate assessments for two kinds of aggregation sets (a) one *system* aggregation set that consists of all system-level metrics (total rate of operations, average latency, etc.) and system-wide resources (e.g. CPU utilization). Also, we generate (b) one aggregation set per *partition*, so that we may compare impact felt by each. In order to derive the statistics given above for a given system, we compare two approaches:

- Use the *system* aggregation set itself as the overall assessment, and
- Combine the *partition* level aggregation sets using a weighted sum, where weights are based on the total number of I/O operations seen by each partition in the assessment duration. This is passed through a percentile threshold to come up with a combined assessment across partitions.

Figure 7 depicts the anomaly detection output for the experiment of Figure 6 with system level aggregation set (first graph

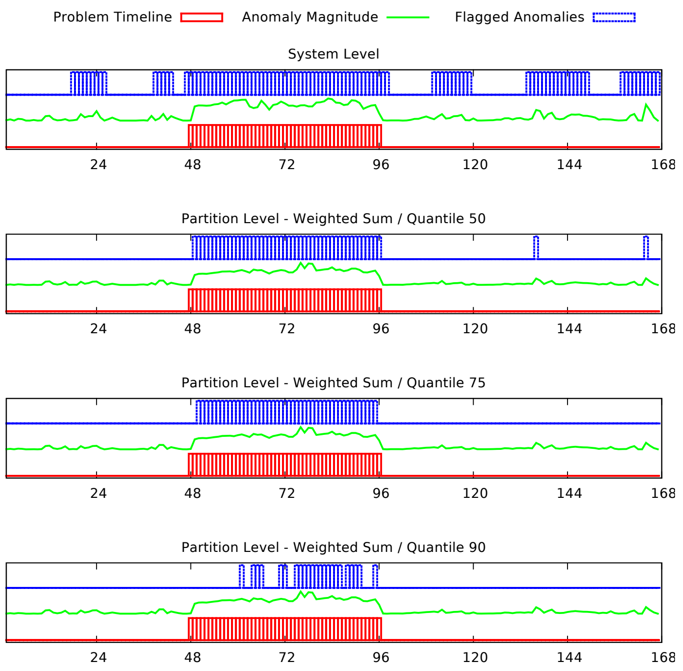


Fig. 7. Sample of anomaly detection with four ways of overall assessment.

in Figure 7) as well as partition-level aggregation with 50^{th}, 75^{th} and 90^{th} percentile filters. Each set of graphs in the figure depicts the problem time line, combined anomaly magnitude, and flagged anomalies detected by *Anode*. As can be clearly seen, a 75^{th} percentile filter on the combined magnitudes (third set of graphs) seems to work better (best visual match between problem time line and flagged anomalies). A more detailed comparison of the statistical metrics defined above for all these assessment methods across all experiments is presented below.

Another useful way to assess the effectiveness of such a methodology is via the well-known *Receiver Operating Characteristic* (ROC) plot. This is a scatter plot between TPR on the y-axis and FPR on the x-axis. A formal description of ROC is left out due to space constraints, but in a nutshell, the ROC curve reveals the TPR attained for a certain amount of FPR. The ideal point for a method to be on this graphs is at the top-left corner where TPR=1 and FPR=0. A curve of TPR vs. FPR is obtained (called the ROC curve) as we vary a threshold to convert anomaly magnitudes into anomaly flags (instead of the definition in Section IV-C). For instance, see Figure 8 for sample ROC curves for four metrics from one experiment (all four repetitions). Here the $y = x$ diagonal represents chance behavior, with curves above diagonal being better. A related statistic is the Area Under ROC Curve (AUC). This is the area enclosed between the ROC curve and the TPR=0 and FPR=1 lines. We use a derived value called Area Above Diagonal (AAD) defined as:

$$\text{AAD} = \text{AUC} - 0.5 \tag{28}$$

This value being positive means better detection of the problem by that metric (Figs. 8(a), 8(b)). If the AAD value is close to zero or negative (Figs. 8(c), 8(d)), it means that the metric does not do a good job of detecting the problem time-periods. We calculated AAD for *each metric* used in the sample experiment above. Figure 9 depicts the spread of AAD values where the x-axis is each metric from the experiment sorted in descending order of their AAD values (note that metric names have been omitted to keep the graph legible). In this manner, one can select the metrics that best indicate a problem (i.e. *symptoms* of the problem).

While ROC requires knowing the ground-truth value, other scoring mechanisms described in Section IV-D2 do not have this limitation. These may be used to find most impacted metrics in any system being analyzed for the first time. Looking across aggregation sets instead of metrics, one can use these scoring mechanisms as ranks for instances of partitions, resources etc. The most impacted partition, would have the highest score, and so on.

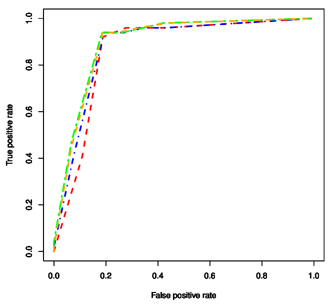
2) Summary across experiments: Moving beyond the sample experiment, let us look at summary stats across the 24 combinations of experiments described earlier. In order to assess *Anode*, we show three types of graphs for statistics gathered across the 24 experiments. These graphs (see Figure 10 for instance) are:

- Summary statistics: Median true positive rate (TPR), false positive rate (FPR), precision and accuracy across the 24 experiments. Ideally, TPR should be 1.0, FPR should be 0.0, precision should be 1.0 and accuracy also should be 1.0.
- Breakup of TPR vs. FPR: Histogram spread of FPR across the 24 experiments (called cases in the graphs). We have binned the TPR and FPR into bins of size 0.25, to get an idea of how the spread looks across the range of FPR observed in the 24 experiments.
- Breakup of FPR vs. TPR: Similar to the previous one, this is a histogram spread of TPR across the 24 experiments.

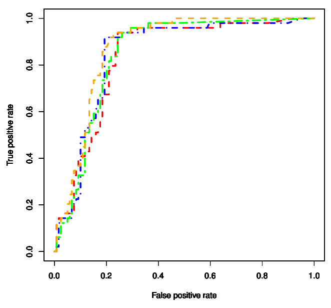
Note that each of the 24 experiments was repeated four times in order to assess the difference between different runs. We found that the error margins for the artificially generated disturbance time-line and the corresponding anomaly magnitude time-line are less than 1% for four runs of the same set of experiments. Since the error margins are negligible we have used results from only one randomly chosen run, out of the four repetitions, to represent each of the 24 experiments in the results here.

The results depicted in Figure 10 present the aggregate stats for 24 experiments across the four types of overall per-experiment assessment, i.e. system-only and partition-weighted-sum with three percentile thresholds (50, 75, 90). Clearly, the system-only assessment (see Figure 10(a)) is not very good even though it has the highest true positive rate (TPR). This is because it has higher false positive rate (FPR), lower accuracy and precision compared to the others. The intuitive explanation for this is that system-level metrics are a superimposition of all the processes/ workloads in the system, and hence have higher false positives when compared to a given problem time-line.

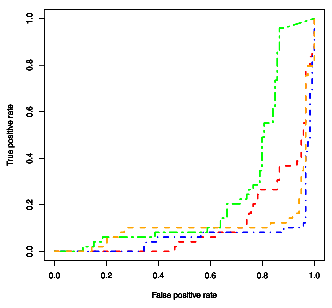
Looking at each partition-level assessment results in b on an hourly basisetter separation of behaviors between workloads that are usually tied to one or more partitions. This is illustrated in the high TPR, low (almost negligible) FPR and high precision and accuracy scores for the partition-weighted-sum based



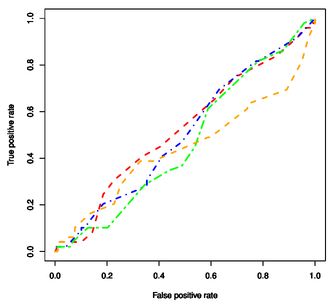
(a) Meta-data operations per second



(b) Latency for meta-data operations



(c) Network data received per second



(d) Write operations per second

Fig. 8. ROC curves for four metrics with (a),(b) showing high area above diagonal (AAD), (c) showing negative AAD, (d) showing close to zero AAD.

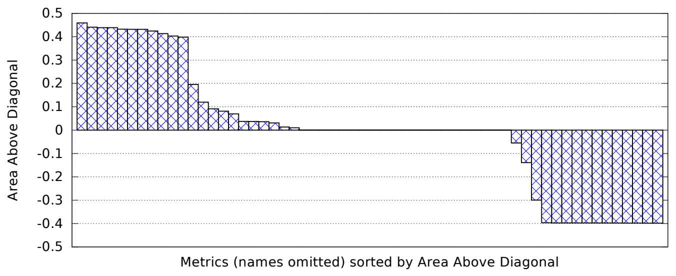


Fig. 9. Ranking metrics by the area (AAD) between the ROC curve and the diagonal value in a single experiment reveals that some metrics (with high positive AAD) are well suited to detecting a problem, while others (with zero/ negative AAD) are ill-suited.

overall assessments in Figure 10(a). The other advantage of using the per-partition assessment approach is that the relative scores (as defined in Section IV-D2) of each partition given by *Anode* allow us to pin-point the most affected partitions. Typically each partition is mapped to a single user-application/ workload, thus aiding the experts in focusing their diagnosis on those applications.

Looking at the histograms of FPR (first column of Figure 10(b)) reveals some interesting observations. Using a relaxed threshold (50^{th} percentile) over the combined anomaly magnitude results in some false positives, and those are limited to some cases where high FPR is observed (second row of Figure 10(b)). Here either most cases have low FPRs (0.0 to 0.25 bin) or very high FPR (0.75 to 1.00 bin). Thus the low (50^{th} percentile) threshold is particularly bad for a significant subset of the 24 experiments. The two higher thresholds (75^{th} and 90^{th} percentile) perform much better in this regard, with all 24 experiments being in the low FPR (0.0 to 0.25) bin as seen in the last two rows of Figure 10(b).

Looking at the TPR histograms (second column of Figure 10(b)) reveals the dual of this behavior. Using a relaxed threshold of (50^{th} percentile) gives very high median TPR (Figure 10(a)), at the cost of some small amount of FPR. TPR is high across the 24 experiments, as depicted in the TPR histogram (right-most graph in the second row of Figure 10(b)). The more conservative thresholds of 75^{th} and 90^{th} percentile have progressively lower median TPR, but zero/ negligible FPR as depicted in Figure 10(a). However, as the TPR histograms (last two rows of Figure 10(b)) reveal, the spread of TPR values has increased (over that of 50^{th} percentile), with a significant number of the 24 experiments falling in the low TPR bin (0.0 to 0.25).

Based on the observations given above, the choice of threshold percentile to use in *Anode* is carried-out as follows:

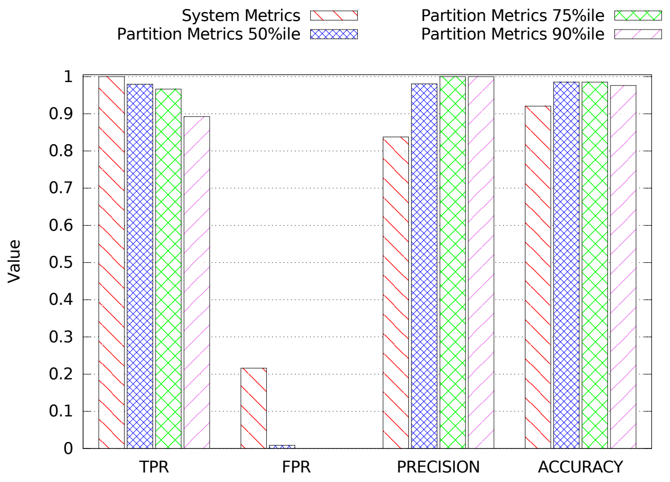
- The high threshold of 90^{th} percentile performs slightly worse overall compared to the medium 75^{th} percentile. Hence the slightly better (with little to no downsides) 75^{th} percentile threshold is preferred.
- Choice between 50^{th} and 75^{th} percentile is trickier, since 50^{th} percentile gives higher TPR at the cost of some FPR, while 75^{th} percentile gives lower TPR, but has the negligible FPR.
- Precision and accuracy are highest for 75^{th} percentile.
- Overall, we select the 75^{th} percentile threshold, to avoid false positives, while still having reasonably high rate of true positives. The intuition being that in a detection and diagnosis setup, having false indications (higher false positives) is more problematic than missing some anomalies (lower true positives).
- The particular threshold numbers were chosen here just for illustration, and this is a parameter configurable by advanced users of *Anode*.

To summarize, the results from our laboratory based study of 24 scenarios shows that *Anode* is able to mark the anomalous periods with high success rate (TPR $> 90\%$) and very low false alarms (negligible FPR). Next, we move to validation over measurement data collected from actual field deployments, which will have much more variability and noise than the controlled environment used up to this point.

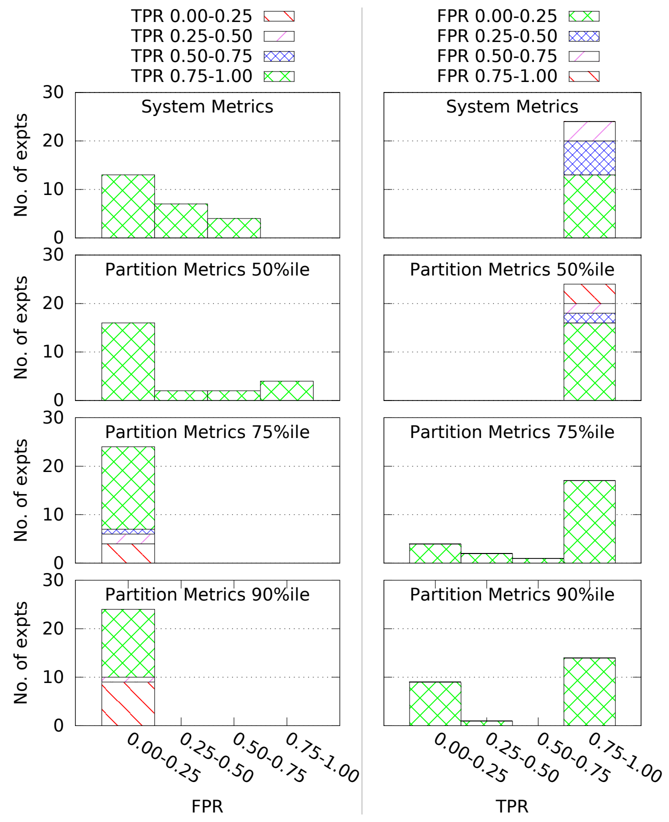
B. Field validation

The laboratory experiment based results showed that *Anode* can catch anomalous durations successfully. Now, one would like to evaluate how well it fares on real performance-related cases filed in the past by customers. However, there is one problem with this study of field data. While *Anode* indicates anomalies in the system on an hourly basis, the corresponding truth value available to us from a field-incident report (case) is the begin and end time. This truth value does not accurately represent what has happened in the given system on a hour-by-hour basis. Three particular problems arise:

- A customer will typically file a case only after they have *noticed* a performance problem. The problem



(a) Summary statistics (median)



(b) Breakup of TPR vs. FPR

Fig. 10. Summary of results from laboratory validation

may have been existing (perhaps to a lesser degree) much before the case is filed.

- The case may remain open for a long time after the actual problem has been solved (e.g. for cross-checks).
- The actual performance impact may be intermittent—coming and going within the duration that a case is open. *Anode* catches this on an hour-by-hour basis, but the same is not reflected in the case-open-close time-line.

Thus there is no way to accurately establish an hour-by-hour truth value for comparison, like we had in the case of laboratory experiments. We work around this limitation by augmenting the case open-close dates in the following manner:

- Anomalies flagged by *Anode* just before the start of the case (within the duration being analyzed) are counted as true positives, since they would indicate existence of the problem before the case was opened.
- Negative outputs (no anomaly flagged) after at least one anomaly has been indicated, but before the close of the case, are counted as true negatives, since these would reflect that the impact has been fixed.

This is as close as we can get to a (so-called) *golden standard* assessment readily for the study of field deployments. A more detailed manual assessment at hourly granularity is sometimes possible, but it would involve far too much expert manpower to create for the purposes of this paper.

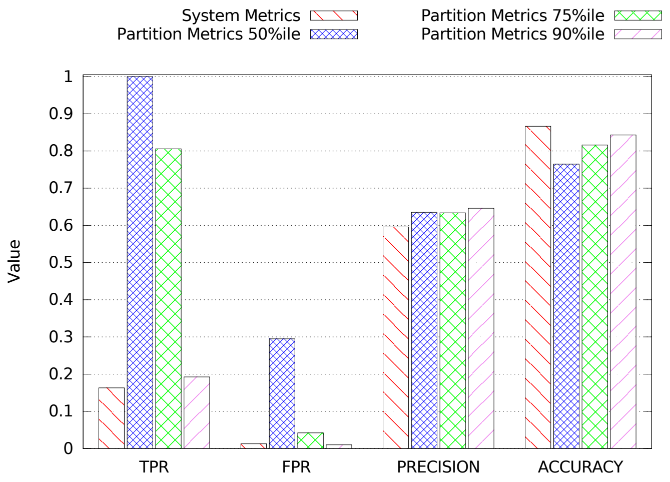
With that done, we can now assess any performance case. For this part we analyze 423 actual cases/ incidents (that were deemed by experts to be performance-related) from actual field-deployed storage systems. The summary of results for this analysis are presented in Figure 11 in the same format as those of the laboratory experiments.

As in the case of laboratory experiments, using only system-wide metrics (see Figure 11(a)) is not very useful. Here it has a low (about 17%) median true-positive rate with very low false-positive rate. This could be usable due to the low false positives, but the lack of true positives is undesirable. Assessments using a weighted sum of individual partition-level assessments are much better even for field deployments. Here weights are based on the number of total operations done over the assessment period. Across all three thresholds depicted (see Figure 11(a)), the 75^{th} percentile threshold shows the best combination of statistics (high TPR, precision, accuracy and low FPR) even in the case of field-validation.

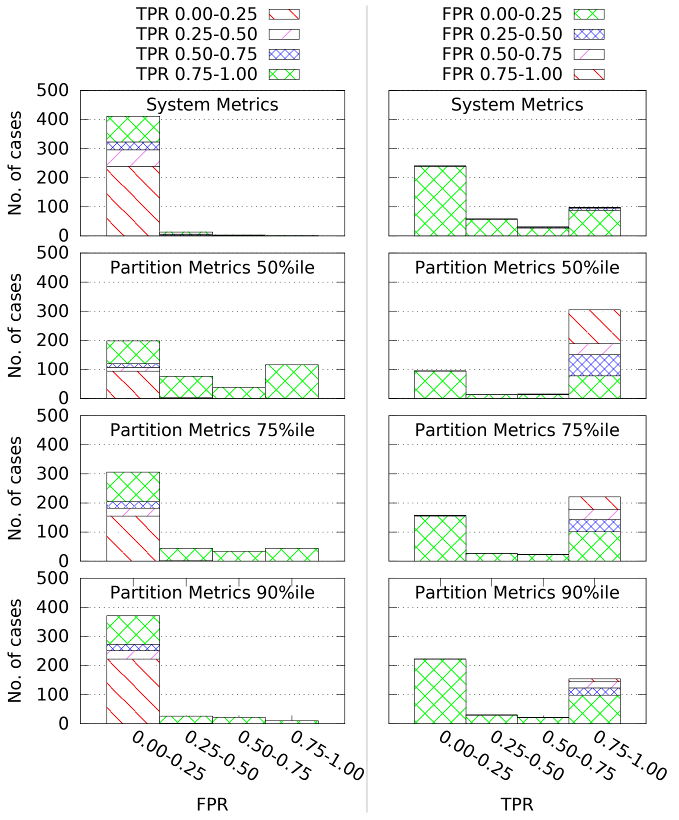
Across the board, the false-positive rate (FPR) is higher in the case of field measurements. This is natural, given that laboratory experiments were done in a controlled environment, but the systems here are serving actual production workloads. Even so, the FPR is not very high, being approximately around the 5% mark (median) for our chosen setup of 75^{th} percentile threshold with partition-level weighted sums (Figure 11(a)). Also recall that this field validation is across 400+ actual cases, compared to merely 24 experiments in the laboratory. Some other observations from the field validation are:

- True-positive rate (TPR) remains high for our chosen setup (75^{th} percentile threshold with partition-level weighted sums) between lab and field results, showing the consistency of the method.
- The slightly higher FPR can be explained as above. This however, also reflects in lower, but very usable accuracy (75% median) and precision (65% median) for our chosen setup (Figure 11(a)).

Overall, the field-data based validation results show good promise in using *Anode* for drilling down into a single case with the use of anomalies, and scoring of metrics, metric sets and other objects. As mentioned in the introduction, focusing the experts attention to the most affected areas could give significant boost to reduction of the time taken to detect and address performance problems. Likewise, automated methods



(a) Summary statistics (median)



(b) Breakup of TPR vs. FPR

Fig. 11. Field validation summary

could be built using *Anode* as the first-level assessment and properly handling possible false-positives at later stages.

VI. Conclusions

In this paper, we have presented a methodology for detecting anomalies in periodic measurements from storage systems. This work is a step towards developing automated problem diagnosis and healing systems for performance related problems. Our method is particularly well-suited to environments with variety of workload patterns given that our method uses a system's own history for determining the baseline for normal behavior of metrics, and can assess on a per-workload basis.

Our method effectively detects the time when a problem impacts metrics, identifies the most affected metrics and most affected parts of the storage system. We have shown the effectiveness of our method on both controlled experiments as well as actual incidents reported from field deployments. Being a data-driven method, it is bound to have some statistically significant probability of making a mistake, but it is still a big improvement over manual or simple rule-based methods—particularly in reducing detection and diagnosis time.

Acknowledgments

The authors would like to thank Kaladhar Voruganti for his help in reviewing several drafts of this work and giving feedback that helped refine the paper. We would also like to thank Swaminathan Ramany, Jason Ledbetter, Ajay Bakhshi, Siddhartha Nandi, Gaurav Makkar, Apoorva Sareen and his team-mates for their guidance and support through the execution of this work.

References

[1] T. Ahmed, M. Coates, and A. Lakhina, "Multivariate online anomaly detection using kernel recursive least squares," in *26th IEEE Intl. Conf. on Computer Communications (INFOCOM)*, 2007, pp. 625–633.

[2] K. M. Carter and W. W. Streilein, "Probabilistic reasoning for streaming anomaly detection," in *IEEE Statistical Signal Processing Workshop (SSP)*, 2012, pp. 377–380.

[3] J. Terrell, K. Jeffay, F. D. Smith, L. Zhang, H. Shen, Z. Zhu, and A. Nobel, "Multivariate SVD analyses for network anomaly detection," in *ACM SIGCOMM Conference, Poster Session*, 2005.

[4] L. Zhang, Z. Zhu, K. Jeffay, J. S. Marron, and F. D. Smith, "Multi-resolution anomaly detection for the internet," in *IEEE Intl. Conf. on Computer Communications (INFOCOM) Workshops*, 2008, pp. 1–6.

[5] R. Fujimaki, T. Nakata, H. Tsukahara, A. Sato, and K. Yamanishi, "Mining abnormal patterns from heterogeneous time-series with irrelevant features for fault event detection," *Statistical Analysis and Data Mining*, vol. 2, no. 1, pp. 1–17, 2009.

[6] S. Das, B. L. Matthews, A. N. Srivastava, and N. C. Oza, "Multiple kernel learning for heterogeneous anomaly detection: Algorithm and aviation safety case study," in *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, 2010, pp. 47–56.

[7] H. Cheng, P.-N. Tan, C. Potter, and S. Klooster, "A robust graph-based algorithm for detection and characterization of anomalies in noisy multivariate time series," in *IEEE Intl. Conf. on Data Mining Workshops (ICDMW)*, 2008, pp. 349–358.

[8] T. H. Lotze, "Anomaly detection in time series: Theoretical and practical improvements for disease outbreak detection," Ph.D. dissertation, University of Maryland, College Park, 2009.

[9] E. Garduno, S. P. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "Theia: Visual signatures for problem diagnosis in large hadoop clusters," in *USENIX Intl. Conf. on Large Installation System Administration (LISA)*, 2012, pp. 33–42.

[10] A. A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao, "Towards automated performance diagnosis in a large IPTV network," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 231–242, Aug. 2009.

[11] W. Yoo, K. Larson, L. Baugh, S. Kim, and R. H. Campbell, "Adp: Automated diagnosis of performance pathologies using hardware events," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1, pp. 283–294, June 2012.

[12] V. Chandola, D. Cheboli, and V. Kumar, "Detecting anomalies in a time series database," Department of Computer Science and Engineering, University of Minnesota, Tech. Rep. TR 09-004, February 2009.

[13] V. Chandola, "Anomaly detection for symbolic sequences and time series data," Ph.D. dissertation, University of Minnesota, 2009.

[14] D. Cheboli, "Anomaly detection of time series," M.S. thesis, University of Minnesota, 2010.