Does the Relationship Between Modules Facilitate in Predicting System Anomaly?

Abstract. In large-scale distributed systems with multiple interconnected modules, failure of even a single module might have a cascading effect and *might* result in overall system failure. Naturally, timely identification and resolution of these system-wide failures is highly crucial for ensuring robust functioning of a system. However, while many frameworks for detecting anomalous behavior exist today, very few models are proposed which can leverage those anomalies for detecting system failures. To fill this gap, in our work, we propose MDAP, a system failure prediction framework for large-scale distributed systems. Specifically MDAP made the following contributions: (a) It detects anomalous behaviour in a system by considering *behavioral changes of the interdependecies* across different modules of the system with only information from system logs. (b) Provides the ability to raise *early* failure alerts. (c) Finally MDAP achieves 39% - 52% better accuracy compared to the baseline algorithms, in identifying periods when the system is performing abnormally —we achieve a high true positive rate of *70.78%* and low false positive rate of *3.56%* using only a few bug information.

1 Introduction

Large scale enterprise systems consist of multiple components operating synchronously as well as asynchronously. Thus, operational breakdown of a particular component may lead to system failure, which affects a huge volume of customers [4, 5]. Unfortunately, specific causes behind system anomalies are hard to diagnose, often involving manual intervention, which increases the cost, both in terms of time and effort. In literature, multiple attempts have been made to develop anomaly detection algorithms by mining the telemetry data from sequential traces as well as analyzing the periodically collected event-driven counters [3, 7, 8, 10] (details in Sec. 2). Clustering based techniques have also been developed for detecting changes in system behaviour to predict anomaly [4–6, 11, 14]. However, most of the state of the art endeavors demand expensive & sophisticated learning infrastructure and are restricted by various statistical assumptions on the system log properties [1, 4, 9, 12].

Notably, large scale enterprise systems are often complex, consisting numerous interconnected sub-systems & modules. Under normal circumstances, those modules exhibit successful coordination among themselves while executing. However, when anomaly sets into a system, some modules may start malfunctioning, which often leads to a cascading effect to other modules, resulting in overall system failure with large downtime. Albeit those module dependencies may lead to

catastrophe, however, we claim that those dependencies, if exploited judiciously, may provide hidden signatures apriori regarding the health of the system. Hence, module dependencies may function as an effective tool to develop an end-to-end anomaly detection framework.

However, obtaining the module dependency information from large scale enterprise system is a challenging task. First, in large scale system, none of the service engineers has the complete view of the entire infrastructure, due to its inherent complexity. Moreover, dependencies among the system modules are not static, which vary depending on the execution environment & time. Close inspection reveals that system modules regularly generate event log messages while executing. By extracting statistics from those raw execution logs, one may compute various attributes of each module (Sec. 3). Hence, the profile of each module can be represented with a tuple of attributes. One may approximate the relationship between two modules by computing the correlation between the respective module attribute profiles. Our pilot study reveals that the correlation of attribute profiles between a module pair exhibit discriminative signatures for normal and abnormal state of the system (Sec. 4).

Motivated from this study, in this paper, we develop an anomaly detection framework MDAP (Module Dependency based Anomaly Prediction), which is equipped to predict impending failures from the system log (Sec. 5). The crux of the model is to discover the hidden relationship between the constituent modules of the system. Since the module relationship is dynamic in nature, we apply ML techniques to discover the relationship between the modules (as weight matrix $\mathbb{WM}_x$) on each day I_x, from their respective attribute profile. Finally, we develop an anomaly prediction framework MDAP leveraging on the discovered weight matrices, for early prediction of anomaly (Sec. 6). We evaluate the performance of MDAP on the event log messages & customer reported cases, crawled from the NetApp storage system (Sec. 7). We observe that MDAP comfortably outperforms all the baseline algorithms, attaining mean accuracy of 81.3% for correctly detecting anomaly, with just around 3.56% false positive alerts (Sec. 8). Side by side, MDAP can predict the anomaly roughly 9 days ahead of the case filing date of the customer.

2 Related work

We provide a brief treatise on the anomaly detection techniques on time-series data in this section which are often leveraged by previous works.

2.1 Log analysis

Most prior works in log analysis focus on the task of mining a log which encloses multiple classes of events related with resource identifiers. These events form sequential traces which are modeled to identify anomalies [3, 7]. Some works also leveraged event driven counters—counters can be collected periodically and helps in better diagnosing system performance issues [8, 10].

2.2 Natural Language Sequence based techniques

Some prior research detected sequential or quantitative anomalies by modeling system logs in a way similar to a natural language sequence [1, 2, 9]. These sequences were designed to detect both sequential and (sometimes) quantitative anomalies using unsupervised approach. Some works even used LSTM (Long Short-Term Memory) on these sequences for anomaly detection [1, 9, 12, 13, 15].

2.3 Clustering based techniques

Clustering and classification based techniques are popularly used in anomaly detection due to their clarity and low latency [6, 11, 14]. In fact some works like EGADS used time series clustering and anomaly detection algorithms to handle specific monitoring use-cases [5]. Another related works proposed ADELE [4]. ADELE attempts to predict system failure by mapping unknown cases given by user to known bug and detect anomaly signatures. The goals of ADELE and EGADS are closest to our problem setting. However, ADELE is based on a score to detect the anomaly which considers (i) only few timeperiods where a bug appeared for producing the score and (ii) assumes that the attribute value must be normally distributed. On the other hand in EGADS the deviation metric is not an optimal metric to detect an anomaly for all time series. Notably, most of these models require an effective learning process necessitating strong assumptions, e.g., normally distributed attribute values or linear relations.

3 Dataset and Problem statement

In this section, we briefly describe the dataset used in our study and define the problem addressed in this paper.

3.1 Data collection infrastructure

In our study, we collected the system generated event logs and command history etc. from the NetApp storage. NetApp provides multiple nodes, which are clustered together in a typical Data ONTAP setup [4]. Individual nodes in the storage system are themselves composed of many interconnected modules, which consistently generate event log messages (EMS log). Every node consists of their own instance of modules such as *wafl*, *raid* etc. Structure of typical EMS log entry is summarized in Table 1.

Side by side, NetApp provides customer support portal facilitating customers to report their grievances as *cases*. Whenever a customer faces certain issue, she files a support case - the case history is logged in a customer support database. We query this database for fields like opened date, closed date, priority label etc for customer support *cases*. The collection of intrinsically related case reports are assigned to a specific bug category (say, performance drop, hardware failure etc) by the service engineers, which is recognized by an unique bug identifier. We use this bug category database to query the individual customer cases associated with a specific bug.

Table 1: EMS MESSAGE STRUCTURE

Field	Log Entry Example	Description
Event Time	Mon Jan 30 07:15:10 PDT	Day, date, timestamp and timezone
System name	zapi	Name of the node in cluster that generated the event
Event ID	snapshot.success.exe	EMS event ID. Contains Subsystem name and event type
Severity	NOTICE	{Severity of the event}
Message String	successful exe00	Message string with argument values

3.2 System model and problem definition

We model the NetApp system as N nodes, where each node $n_i \in N$ is composed of k modules $\mathbb{M} = \{m_1, m_2 \ldots, m_k\}$. Each module m_i generates an EMS log, which we represent as attribute tuple $\mu_i = \{x_i, y_i, p_i, q_i\}$. For a module m_i, *attributes* x_i, y_i, p_i, q_i represent event count, event ratio, mean inter-arrival time, mean inter-arrival distance respectively [4]. We define **event count (EC)** of a module m_i as the number of event messages generated by that module. **Event ratio (ER)** is the fraction of events generated by the module m_i. **Mean inter-arrival time (MT)** is the mean time between successive events of the module m_i. **Mean inter-arrival distance (MD)** is defined as mean number of events by other modules between successive events of module m_i. For each bug $B^y, y \in [1:5]$, we crawl the EMS log history of C^y different cases, where $C = |C^y|$ depicts the number of cases in bug B^y. For a case $c_j \in C^y$ generated from the node $n_i \in N$, we collect the EMS log for D number of days, with one day logging interval. Precisely, for each day I_x, we collect the EMS log L_x^j ($x \in [1:D]$) from the case c_j. Each EMS log of day I_x, $L_x^j = \{\bigcup \mu_i^j, i \in [1:k]\}$ represents a vector of dimension k. If a customer files a case report for c_j on day I_c, observing some anomaly in the node n_i, we label all the prior days $I_f, \forall I_f \leq I_c$ as normal and all the subsequent days $I_f, \forall I_f > I_c$ as abnormal. Now consider that for a node n_i on day I_p, we obtained the EMS log L_p^h from a new case c_h. We aim to develop a framework which is able to predict if the the node n_i on day I_p is in normal or in abnormal state. In order to develop the framework, we rely on all the EMS log L^i collected from the past cases $c_i \in C_y$, as well as available EMS log collected from the target case c_h.

4 Challenges and Opportunity

Major state of the art algorithms ADELE [4] and EGADS [5] primarily rely on the specific attribute values from raw execution logs, obtained from the system modules. A favorable situation for these two methods are shown in Figure 1 for the event count attribute for two modules in our dataset. Note that the anomalies are represented as sudden spikes in the attribute values; both ADELE and EGADS perform well in those simplistic scenarios. However, those methods face multiple challenges.

4.1 Limitations of existing approaches

(a) Presence of sporadic spikes: Both ADELE and EGADS consider radical variations in attribute values (spikes) over time as an indication of anomaly in

the system. While this might be true for some modules, other modules are quite likely to produce sporadic attribute values even during normal or non-anomalous periods, as shown in Figure.1(a). Moreover, some modules show gradual increase in attribute values in case of impending failures. Hence naively relying on the attribute values as anomaly indicator may often result in false positives.

(b) Assumption of normal distribution: ADELE calculates the anomaly score by observing the changes in an *attribute* value over past few days, for all the k modules. Precisely, ADELE assumes that an attribute (say, event count), obtained from all k modules, follow *normal distribution*. On a specific day log, any deviation from that distribution is flagged as anomaly via outlier detection technique. However, this is a strong assumption, specifically for small sample of days, where attributes may fail to follow normal distribution, resulting in incorrect predictions.

4.2 Opportunity for MDAP

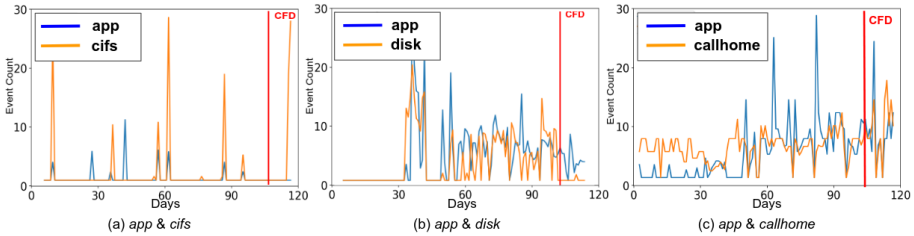


Fig. 1: Module pair correlations (in terms of EC) with time. CFD indicates case file date.

We address these limitations in MDAP with a simple observation. In large enterprises, often a single module failure leads to a cascading failure of other modules present in the system. We conceptualize this cascading effect with the help of correlation in attribute values present within the module pairs; any deviation in correlation should hint at impending failures.

We present an anecdote to validate this hypothesis with three pairs of modules; *app* & *cifs* & *app* & *disk* and *app* & *callhome* (see Figure 1). We plot the attribute (event count) values obtained from these modules over a period of time. First of all, in Figure 1, we observe that module pair correlations exist for at least *some* period of time. However, closer look at Figure 1) reveals that existing correlations get disrupted during an anamolous period (Figure 1(b)), whereas few new module-pair correlations start appearing with an impending failure (see Figure 1(a) Figure 1(c)). These observations motivate us to systematically leverage the relationship between the various modules to identify impending system anomaly.

5 Development of MDAP

In this section, we illustrate the steps to develop MDAP for anomaly prediction. MDAP is essentially as suite of models $\{W^y_{EC}, W^y_{ER}, W^y_{MT}, W^y_{MD}\}$, developed

based on each module attributes event count(EC), event ratio(ER), mean inter-arrival time(MT), mean inter-arrival distance(MD) respectively. This is important to note that the each model is bug specific, hence we develop one single model W^y_{EC} for a all the cases C^y of a specific bug B^y.

5.1 Construction of feature matrix

We construct the feature matrix $\mathbb{FM}_x$ for each day $I_x \in D$, based on the EMS log collected from all the cases in C^y. We collect the EMS log L^j_x from a specific case $c_j \in C^y$ of day I_x. We represent the log L^j_x as a 1D vector of dimension $1 \times k$ (k is the number of modules) . Each entry of the vector depicts a attribute tuple $\mu^j_i = \{x_i, y_i, p_i, q_i\}$, obtained from the module M_i for case c_j on day I_x, where x_i, y_i, p_i, q_i represents event count, event ratio, mean inter-arrival time, mean inter-arrival distance respectively of case c_j . Next, we collate the log L^l_x of all the cases $c_l \in C^y$ of day I_x and construct the feature matrix $\mathbb{FM}_x = \bigcup_j L^j_x$ of dimension $C \times k$, where $C = |C^y|$ cases present in a bug B^y.

5.2 Learning weight vectors

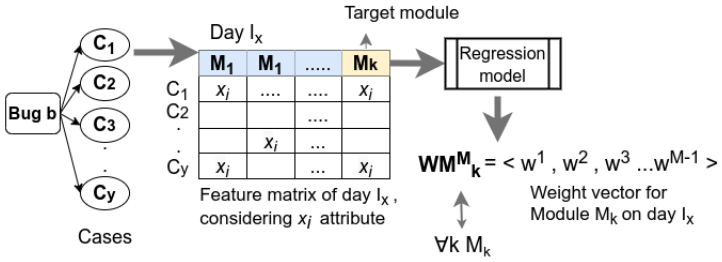


Fig. 2: Weight vector learning to represent module pair relationship.

Once we construct the feature matrix $\mathbb{FM}_x$ for day I_x, we aim to learn the weight vector $\mathbb{WM}^{M_I}_x\ \forall M_i \in \mathbb{M}$ for day I_x. This weight vector $\mathbb{WM}^{M_i}_x$ essentially represents the relationship between one module M_i with $k-1$ other modules and we compute $\mathbb{WM}^{M_i}_x\ \forall M_i \in \mathbb{M}$ to construct weight matrix $\mathbb{WM}_x$. In order to construct the weight vector $\mathbb{WM}^{M_i}_x$ of dimension $1 \times (k-1)$, we concentrate on one module M_i of the feature matrix $\mathbb{FM}_x$ (essentially one column of $\mathbb{FM}_x$). We construct an auxiliary matrix $\mathbb{WM}^{A_i}_x = \mathbb{FM}_x - M_i$ of dimension $C \times (k-1)$ by excluding the column (module) M_i. Next, we take one attribute (say, event count x_i) of module M_i in consideration. We implement a Ridge regression model to estimate the event count attribute x_i of the module M_i from the auxiliary matrix $\mathbb{FM}^{A_i}_x$. Precisely, we train the regression model for a case (row) C_i of the auxiliary matrix $\mathbb{FM}^{A_i}_x$, with the event count attribute x_j, $j \forall [1 : k-1]$ of all other modules to estimate the attribute $\hat{x_i}$ of the module M_i. Ridge regression aims to minimize the loss between the true event count attribute value x_i and the estimated attribute $\hat{x_i}$. We perform learning using Ridge regression which minimizes loss as it maintains the same overall degree of variation. Also It is

most suitable when a data set contains a higher number of predictor variables (k) than the number of observations (C). We repeat this procedure for all the cases $c_i \in C_y$ and in the process, we learn the weight vector $\mathbb{WM}_x^{M_i}$ of dimension $k-1$ for day I_x. We repeat this procedure for each module $M_i \in \mathbb{M}$ to learn k weight vectors $\mathbb{WM}_x^{M_i}, \forall M_i \in \mathbb{M}$. Finally, we obtain the weight matrix for day I_x as $\mathbb{WM}_x = \bigcup_{M_i} \mathbb{WM}_x^{M_i}$ of dimension $k \times |k-1|$. In this way, for each day $I_x \in D$, we construct a weight matrix W_x^y and for all the D days, we construct the model $W_{EC}^y = \bigcup_{x \in D} \mathbb{WM}_x$ for bug y. Overview of the procedure has been explained in Figure 2. Notably, we constructed the aforesaid model W_{EC}^y focusing on the *event count* module attribute. In similar line, we consider the attributes event ratio (ER), mean inter-arrival time (MT), mean inter-arrival distance (MD) and construct the respective models W_{ER}^y, W_{MT}^y, W_{MD}^y.

6 Anomaly Prediction

We aim to utilize the proposed framework to diagnose if the node n_i is in anomalous state on day I_p from the collected EMS log L_p^j. This log L_p^j is represented as a 1D vector of dimension $1 \times k$. Each entry of the vector depicts a attribute tuple $\mu_i^p = \{x_i, y_i, p_i, q_i\}$, obtained from the module M_i, where x_i, y_i, p_i, q_i represents event count, event ratio, mean inter-arrival time, mean inter-arrival distance respectively. In the following, we focus on the event count x_i, and apply the corresponding model W_{EC}^y. However, the similar procedure can be repeated for other attributes as well, considering the respective models W_{ER}^y, W_{MT}^y, W_{MD}^y. The anomaly detection is performed in two steps. First, we detect the abnormality in the system. Notably, abnormality does not always lead to system anomaly. In the second step, we predict the anomaly in the system.

6.1 Detecting system abnormality

First, we construct two classes of weight matrices (i) normal ($\mathbb{WM}_N$) and (ii) abnormal ($\mathbb{WM}_A$), from the the proposed model W_{EC}^y, where weight matrix $\mathbb{WM}_N = \{\mathbb{WM}_x | I_x \textit{ is normal}\}$ and $\mathbb{WM}_A = \{\mathbb{WM}_x | I_x \textit{ is abnormal}\}$. The normal (or abnormal) class $\mathbb{WM}_N$ (or $\mathbb{WM}_A$) contains all the weight matrices $\mathbb{WM}_x$, whose corresponding days are labelled as normal (or abnormal). Next, we pick one day $I_t \in D$ and select the corresponding weight matrix $\mathbb{WM}_t$ from the MDAP W_{EC}^y. This weight matrix is essentially a collection of weight vectors $\mathbb{WM}_t^{M_i}$ of dimension $1 \times k-1$, one for each module M_i. We start with one module M_i, and note its event count x_i from the log L_p. Now we aim to estimate the event count $\hat{x_i}$ of the module M_i from the event count x_j of other (k-1) modules $M_j \in [\mathbb{M} - \mathbb{M}_i]$. We apply the learnt weight vector $\mathbb{WM}_t^{M_i}$, obtained from W_{EC}^y, on the event counts $x_j, \forall M_j$ of (k-1) modules to estimate the event count $\hat{x_i}$ of module M_i. We compute the error in estimation $e_t^i = |x_i - \hat{x_i}|$ for module M_i. We repeat the process for all the k modules to compute the mean error $\hat{e_t} = \sum_{i \in \mathbb{M}} e_t^i$ for day I_t. We repeat this procedure for all the $I_t \in [1:D]$ days to obtain the error vector $E = \{\hat{e_t} | t \in D\}$ and find the day I_w which shows the minimum error as $I_w = \text{argmin}_t\, f(E = \{\hat{e_t} | t \in D\})$. That day I_w and the

corresponding weight vector $\mathbb{WM}_w \in W^y_{EC}$ is the key indicator of abnormality. If that weight matrix $\mathbb{WM}_w \in \mathbb{WM}_N$, we predict that the log L_p of day I_p as normal. Otherwise, if the weight matrix $\mathbb{WM}_w \in \mathbb{WM}_A$, we predict that the log as abnormal.

6.2 Predicting Anomaly

Given an EMS log L_p collected from day I_p, the proposed framework W^y_{EC} detects an *abnormality* present on day I_p. However, a mere abnormality does not always reflect a permanent *anomaly* in the system, which may lead to system failure. Often such abnormality gets automatically disappeared in a self healing manner. Hence, the proposed framework declares anomaly for an EMS log L_p, collected on day I_p, if W^y_{EC} detects abnormality in the EMS log collected on day I_p, as well as in the log collected in *all* prior λ days. Notably, λ is a system parameter and should get tuned based on the environment. In all our experiments, we fix the $\lambda = 3$ days.

7 Experimental setup

We conduct the evaluation of the proposed framework MDAP on the collected NetApp dataset, consisting $N = 400$ number of nodes and total $k = 331$ modules. We concentrate on 5 unique bugs with on average 90 cases. For each case, we crawled the EMS log of total $D = 126$ days, where we label 12 days before and 14 days after the case file date as *abnormal* days (since anomaly slowly sets into the node, even before customer notices it). Rest 100 days are labelled as *normal* [4].

7.1 Model training & evaluation procedure

We construct the model W^y_{EC} for each bug B^y and implement the leave one out cross validation to evaluate its performance on the cases $c_j \in C^y$. During evaluation, we keep one case c_j as hold out, and train the Ridge regression model on the EMS log of other $C^y - c_j$ cases for the D days to learn the weight vector $W^{y_j}_{EC}$. Additionally, in order to capture the case specific information, we also include T days log of the test case c_j to train $W^{y_j}_{EC}$. Once we construct the model $W^{y_j}_{EC}$ for the test case c_j, we utilize the EMS log samples of rest $D - T$ days of case c_j to evaluate the model. Note, this testing days $D - T$ include both normal and abnormal days. We repeat this procedure for each test case $c_j \in C_y$ and report the average model performance validating with ground truth. Since the dataset is unbalanced containing more normal days compared to abnormal days, we perform undersampling to balance the data while executing the Ridge regression model.

7.2 Baseline algorithms

We implement the following baseline algorithms to evaluate the performance of MDAP. **(a) EGADS [5]:** This is an open-source Java package to automatically

detect anomalies in large scale time-series data. It implements time-series algorithms to compute an expected anomaly at a given time, which has three different variation namely KSigma, DBScan and ExtremeLowDensity. **(b) ADELE Direct/ADELE [4]:** This model learns from system's own history to establish the baseline of normal behavior and observes the module attribute deviations from normal to flag anomaly. **(c) Aggregated MDAP:** We implement a variation of MDAP as baseline, where we construct a single aggregated model for all the five bugs. Here we simply rely on all the cases obtained from different bugs while learning the weight vectors in the regression process.

8 Evaluation

In this section, we conduct rigorous performance evaluation of MDAP.

8.1 Overall performance

In Table 2b, we evaluate the framework MDAP = $\{W^y_{EC}, W^y_{ER}, W^y_{MT}, W^y_{MD}\}$ on two popular bugs. We observe that all the models $W^y_{EC}, W^y_{ER}, W^y_{MT}, W^y_{MD}$, developed based on various module attributes, achieve pretty high accuracy and they are close to each other. Moreover, this is comforting for us to observe that MDAP exhibits low false positive rate, reducing the false alerts for the support engineers. However, since the model W^y_{EC} exhibits highest accuracy, for brevity, we concentrate on W^y_{EC} in the rest of the section as a proxy of MDAP.

Table 2: Performance of MDAP

Bug Id.	**Accuracy**	**FP**
519766	85.4%	3.98%
531755	81.3%	2.52%
787560	81.5%	1.7%
817812	79.7%	4.56%
822180	78.8%	4.98 %
Avg.	**81.34%**	**3.56%**

(a) Bug wise accuracy

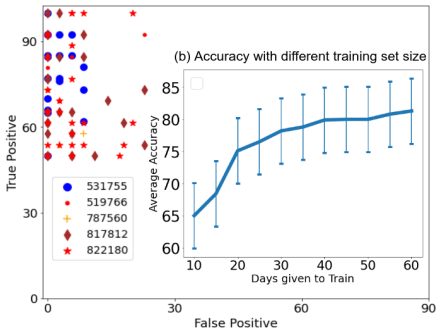
Model Used	**Bug 531755**	**FP**	**Bug 817812**	**FP**
W^y_{EC}	81.3%	3.9%	79.7%	4.2%
W^y_{ER}	77%	4.81%	72.7%	7.44%
W^y_{MD}	75.2%	7.02%	73.8%	5.7%
W^y_{MT}	77.9%	5.51%	79.9%	4.1%

(b) Accuracy with different MDAP models

Table 2a shows the bug wise accuracy obtained by MDAP, where we achieve an average accuracy of 81.3%, with low average false positive rate 3.56%. This result is further illustrated in Figure 3a, where each point in the scattered plot depicts a case of a bug. The position of the point in the scattered plot depicts the TPR (in Y-axis) and the FPR (in X-axis) of that case. The concentration of the points at the top left corner substantiates the elegance of MDAP achieving high accuracy with low false alerts.

8.2 Baseline comparison

Table 3b shows that MDAP comfortably outperforms all the baseline algorithms. The poor performance of baseline stems from the fact that ADELE [4]



(a) Scattered plot showing True Positive Rate vs False Positive Rate.

Model	Average Accuracy	TPR	FPR
EGADS Ksigma	27.19%	61.7%	79.4%
EGADS DBSCCAN	50.14%	37.41%	46.19%
ADELE DIRECT	49.58%	37.4%	46.1%
ADELE	58.4%	61.3%	41.8%
Aggregated MDAP	30.3%	18.8%	83.7%
MDAP	**81.3%**	**70.78%**	**3.56%**

(b) Comparison with baseline algorithms

Fig. 3: Demonstrating the elegance of MDAP

relies only on the outlier detection from the module attribute distribution (with a strong assumption of normal distribution). Similarly, EGADS [5] predicts anomaly only from the time-series dependencies of module attributes. However, MDAP learns the correlation between the modules (as weight vectors) and correctly discovers its relationship with the system anomaly. The performance of Aggregated MDAP drops substantially as this model fails to capture the intrinsic variation present across the cases of different bugs.

8.3 Dissecting MDAP

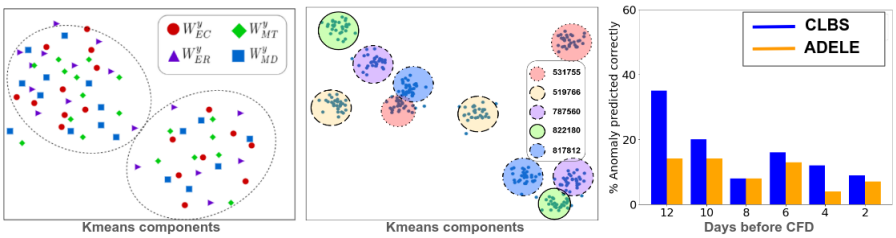
Finally, we dissect MDAP to uncover various interesting insights.

(a) Early Prediction of anomaly: In Figure 4(c) we show the early prediction ability of MDAP. On the x-axis, we plot the number of days an anomaly was predicted, before the case file date, and the y-axis shows the fraction of such correctly predicted cases. We observe that MDAP can in most of the cases set in warning much before (on average **9.3** days) actual failure sets in MDAP, which is better then baseline framework ADELE (on average **5.2** days).

(b) Impact of training volume T: In order to capture the case specific properties of the test case, we include T days EMS log to train MDAP. Figure 3a (inset) shows that performance of MDAP sharply improves with training volume x. Notably, with just 30% training data from the test case, MDAP achieves near 80% accuracy.

(c) Anomalous vs normal weight matrices: Figure 4(a) & 4(b) confirms that the normal class weight matrices $\mathbb{WM}_N$ are intrinsically different from the abnormal class $\mathbb{WM}_A$. In this scattered plot, each point represents one weight matrix $\mathbb{WM}_x$ computed with $^{y}_{EC}$ model (Fig. 4(b)). We observe that for each bug,

the weight matrices in the normal class are concentrated in one location, which is positionally different from the concentration of the abnormal class weight matrices. Figure 4(a) extends this notion for all the models $W^y_{EC}, W^y_{ER}, W^y_{MT}, W^y_{MD}$. This plot demonstrates the rationale behind the superiority of MDAP in discriminating normal and abnormal EMS logs.



(a) Weight Clustering with different attributes (Bug 531755)

(b) Weight Clustering Bug wise (W^y_{EC})

(c) Early Anomaly Prediction Comparison

Fig. 4: Properties of weight matrices and Early prediction of anomaly

9 Conclusion

In this paper, we have developed an anomaly detection framework MDAP, which leverages on the relationship between the constituent system modules. Our pilot study revealed that relationship between the constituent module pairs sharply changes, once the anomaly sets into the system. We rely on this change as an indicator to discriminate between the normal and anomalous state of the system. We apply machine learning to discover the relationship between the modules, in the form of weight vectors, based on the EMS logs collected from the past cases. This notion of module pair relationship facilitates MDAP to attain an average accuracy of 81.3% for correctly detecting anomaly, with just around 3% false positive alerts. Side by side, MDAP can identify anomaly in the system roughly 9 days early. This essentially means that if support engineer can intervene in any one of those 9 days, the impending failure can be avoided. We claim this as a remarkable advancement compared to the performance observed by the state of the art baseline algorithms.

References

1. Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298, 2017.
2. Okwudili M Ezeme, Qusay H Mahmoud, and Akramul Azim. Dream: deep recursive attentive model for anomaly detection in kernel events. *IEEE Access*, 7:18860–18870, 2019.
3. Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*, pages 149–158. IEEE, 2009.

4. Subhendu Khatuya, Niloy Ganguly, Jayanta Basak, Madhumita Bharde, and Bivas Mitra. Adele: Anomaly detection from event log empiricism. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2114–2122. IEEE, 2018.
5. Nikolay Laptev, Saeed Amizadeh, and Ian Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1939–1947, 2015.
6. Zhaoli Liu, Tao Qin, Xiaohong Guan, Hezhi Jiang, and Chenxu Wang. An integrated method for anomaly detection from massive system logs. *IEEE Access*, 6:30602–30611, 2018.
7. Leonardo Mariani and Fabrizio Pastore. Automated identification of failure causes in system logs. In *2008 19th International Symposium on Software Reliability Engineering (ISSRE)*, pages 117–126. IEEE, 2008.
8. Vipul Mathur, Cijo George, and Jayanta Basak. Anode: Empirical detection of performance problems in storage systems using time-series analysis of periodic measurements. In *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–12. IEEE, 2014.
9. Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *IJCAI*, volume 19, pages 4739–4745, 2019.
10. Vinod Nair, Ameya Raul, Shwetabh Khanduja, Vikas Bahirwani, Qihong Shao, Sundararajan Sellamanickam, Sathiya Keerthi, Steve Herbert, and Sudheer Dhulipalla. Learning a hierarchical monitoring system for detecting and diagnosing service issues. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2029–2038, 2015.
11. Chitranshu Raj, Lavanya Khular, and Gaurav Raj. Clustering based incident handling for anomaly detection in cloud infrastructures. In *2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pages 611–616. IEEE, 2020.
12. Ruipeng Yang, Dan Qu, Ying Gao, Yekui Qian, and Yongwang Tang. Nlsalog: An anomaly detection framework for log sequence in security management. *IEEE Access*, 7:181152–181164, 2019.
13. Steven Yen, Melody Moh, and Teng-Sheng Moh. Causalconvlstm: Semi-supervised log anomaly detection through sequence modeling. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 1334–1341. IEEE, 2019.
14. Xiao Zhang, Fanjing Meng, and Jingmin Xu. Perfinsight: A robust clustering-based abnormal behavior detection system for large-scale cloud. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 896–899. IEEE, 2018.
15. Pengpeng Zhou, Yang Wang, Zhenyu Li, Xin Wang, Gareth Tyson, and Gaogang Xie. Logsayer: Log pattern-driven cloud component anomaly diagnosis with machine learning. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2020.