Coordinated Analysis of Heterogeneous Monitor Data in Enterprise Clouds for Incident Response

Uttam Thakore*, Harigovind V. Ramasamy†, William H. Sanders*
*University of Illinois at Urbana-Champaign, Urbana, IL 61801
†IBM, Austin, TX 78758
Email: {thakore1, whs}@illinois.edu, {hvramasa}@us.ibm.com

***Abstract*— During incident analysis and response, enterprise cloud administrators want to use as much of their generated monitor data as possible. However, the reality is that decisions are often dictated by the tools actually available to automatically process the monitor data, rather than by an understanding of the relevance of the data for incident response. The significant manual effort and domain expertise required to process diverse cloud monitors means that much monitor data remain unexamined. We propose a framework for simplifying the complexity of data analysis for incident response. Our framework enables coordinated analysis of both metric (numerical) data and log (semi-structured, textual) data and exposes salient features within those data. As a foundation for the framework, we define a taxonomy for fields within monitor data based on insights gained from analyzing logs and metrics collected from all levels of an experimental platform-as-a-service (PaaS) cloud (EPC). Using the taxonomy, we lay out a method for semi-automated feature extraction and discovery across heterogeneous monitors. We then describe a method for feature clustering to promote effective analysis of the data, and to remove redundant and uninformative features. We discuss the application of our framework for incident response within the EPC, including root cause analysis.**

***Index Terms*—cloud computing, log analysis, reliability, incident response, log clustering, AIOps**

I. Introduction

In large-scale enterprise clouds, large volumes of system data are collected for detection and investigation of undesirable incidents, such as failures, performance bottlenecks, and attacks. As cloud platforms grow in size and complexity, the diversity and quantity of monitors that can be deployed to collect data about system operation for incident analysis, as well as the number of features that can be extracted from the data, increase dramatically [1]. It is not always possible to efficiently analyze information from all monitors that can be deployed, nor is it necessarily known a priori which features will be important in detecting incidents that will occur during system operation.

Existing monitoring and analysis frameworks can automatically parse and analyze data from only a limited set of cloud monitors, and most segregate analysis of numerical and text data, limiting the types of analyses that can be done. Furthermore, existing tools provide limited guidance about which features are salient for reliability and security incident response. For a cloud provider collecting data from multiple monitors, most features are redundant or uninformative, often increasing incident response time. Downtime due to incident response can result in lost revenue and customer dissatisfaction.

In this paper, we introduce a framework to semi-automatically process heterogeneous monitor data from multiple levels of a cloud platform into a manageable set of meaningful time series features useful in incident root cause analysis. We propose a general taxonomy for monitor data fields that administrators can use to easily label both structured and unstructured components of monitor data. We then present a method to automatically extract time series features based on labels from our taxonomy, remove uninformative features, and reduce the overall number of features by clustering together related and redundant features.

Our framework enables *coordinated analysis* of both metric (numerical) data and log (semi-structured, textual) data, which to the best of our knowledge is not supported by existing techniques or tools, and typically presents a challenge to cloud support engineers. Our framework also (1) supports feature extraction from arbitrary cloud monitor types, (2) requires limited annotation or custom parsing by domain experts to operate on parameters within unstructured log messages, (3) aids in the discovery of meaningful features while minimizing feature redundancy, and (4) identifies meaningful relationships between features that can aid in incident response. We demonstrate the utility of our framework on incident data collected from an experimental PaaS cloud service (EPC), which was serving as a development-test environment. Our approach significantly reduces the manual effort needed of administrators to ingest and operationalize monitor data.

The rest of this paper is organized as follows. In Section II, we describe the EPC for which we developed our framework and the dataset we use for evaluation. In Section III, we explain how monitoring and incident analysis are performed in practice and illustrate the challenges faced in the EPC. Section IV details our taxonomy of cloud monitor data fields, and Section V describes our procedures for feature extraction and clustering. Finally, in Section VI, we illustrate our approach's ability to aid in feature discovery and incident analysis.

II. System and Data Description

To evaluate our work, we applied our approach to a case study dataset containing three performance and reliability incidents from end-to-end testing of the EPC.

At the base level, each instance of the EPC consisted of three Linux VMs networked together: a web server, an application server, and a database server. The case study dataset contains system and application log and metric data for three separate test failure incidents, each with a different cause of failure. The exact set of monitor data differs for each incident; a summary of the relevant monitor types is given in Table I.

To illustrate the feature extraction challenges faced by administrators of the EPC, we provide sample log lines from the Web application server (WAS) error logs (Listing 1) and the Web application Security Access Manager (WSAM) error logs (Listing 2). The WAS responds to application requests from the web server and submits requests to the database server, so its error logs contain exceptions raised by the application and can provide insight into both application and database

TABLE I: Monitor types available per incident in our PaaS dataset

Monitor type	1	2	3
OS performance metrics (nmon) (all 3 VMs)	✓	✓	✓
Per-process perf. metrics (top) (all 3 VMs)	✓	✗	✗
Linux syslog (web server VM only)	✓	✗	✗
Apache web server access logs	✓	✓	✗
Apache web server error logs	✓	✓	✗
Appl. server error logs	✗	✓	✗
Appl. Security Access Manager request logs	✗	✗	✓
Appl. Security Access Manager error logs	✗	✗	✓

issues. The WSAM authenticates clients of the web server and interacts with both the web server and WAS. Each of the logs contains structured and unstructured fields, where the unstructured messages can span multiple lines and are of variable length, complicating automated log analysis.

III. Background and Challenges

In practice, monitor data are conventionally classified as either *metric data* or *log data* [2] [3]. At a high level, periodically sampled numerical measurements are considered to be metrics, and records of discrete events, whether structured or unstructured (i.e., textual), are considered to be logs. For example, host-level resource utilization values, such as those generated by the Linux top utility, are considered metrics, and logging messages generated by applications, such as those accumulated by the Linux syslog utility, are considered logs.

Our work is motivated in part by the fact that existing log analysis approaches are specialized to individual log types, and cannot be easily adapted to operate on the wide range of heterogeneous data that are present in large-scale cloud systems. Some of the tools more commonly used in practice to process and analyze diverse log and metric data include Microsoft Azure Kusto [4], GrayLog [5], and Splunk [6]. Such tools are sometimes called AIOps (artificial intelligence for IT operations) tools if they use machine learning to process the data. Each of the tools can consume a variety of logs by using community-provided input plugins. However, in most cases, the text components of log messages are not parsed by the plugins because maintainance of the complex regular expressions required for parsing is intractable, as logging statements are constantly being updated [1] [7]. That prevents utilization of parameter values present in the messages, even though those are often important, as we show later in this paper.

Further, logs and metrics are traditionally stored and analyzed separately. Within the EPC, metrics were stored in a Graphite [8] time series database and used to generate dashboards that administrators could monitor in real time, and to generate e-mail alerts when certain undesirable conditions were met. Logs were parsed and indexed using the ELK stack [9], after which some logs were used to create dashboards in Kibana [9] and some were filtered for error or failure keywords and used to generate e-mail alerts.

That dichotomy between log analysis and metric analysis caused a number of problems. First, analysis was fragmented; since system administrators were required to maintain separate dashboards for log and metric data, it was not possible to directly compare the two, so investigation of incidents required one to switch back and forth between dashboards and matching

```
0 [XX/XX/XX ##:##:26:266 UTC] 0000000a com.ibm.ws.session.
      WASSession                E SESN0042E: An error occurred
      when trying to insert a session into the database.
1 [XX/XX/XX ##:##:29:384 UTC] 0000000b com.ibm.ws.webcontainer.util
      .ApplicationErrorUtils     E SRVE0777E: Exception thrown by
      application class 'org.apache.jasper.runtime.PageContextImpl
      .handlePageException:1234'
2 com.ibm.websphere.servlet.error.ServletErrorReport: ERROR: An
      unknown or unexpected error occurred when requesting [...]
```

Listing 1: Sample Web application server (WAS) log lines.

```
0 201X-XX-XX-##:##:56.087+00:00I----- 0x38AD54BA webseald WARNING
      wiv ssl SSLConnection.cpp 860 0x0000000a
1 DPWIV1210W   Function call, gsk_secure_soc_init, failed error:
      0000019e GSK_ERROR_BAD_CERTIFICATE.
2 201X-XX-XX-##:##:24.414+00:00I----- 0x132120DD webseald WARNING
      ias authsvc pdauthn.cpp 2233 0x0000aaaa
3 HPDIA0221W   Authentication for user bob@a.com failed. You have
      used an invalid user name, password or client certificate.
```

Listing 2: Sample Web Security Access Manager (WSAM) log lines.

alert timestamps. Similarly, since the data were stored in different databases in different formats, there was no way to query log and metric data conjunctly.

Second, log data were heavily under-utilized. In order to parse log messages, engineers needed to construct complex regular expressions. Some applications produce highly structured and detailed request and error logs, while others produce only logs with limited structured information (such as message severity level and error code) and predominantly unstructured information, which was typically examined only during manual incident analysis. That left important information unused, especially within parameter values embedded in log messages.

Importantly, the dichotomy between log and metric data was artificially imposed by the tools available to collect and analyze the data, rather than by how the data needed to be used during incident response. In many instances, administrators treated logs like "metrics." For example, the frequencies of different Apache response status codes were counted per 30-minute interval and monitored using a Kibana dashboard, as were the frequencies of the application server error log messages at ERROR and FAILURE severity levels. The admins needed a coordinated method to convert the data from different monitors in the EPC into a meaningful set of time series features that they could easily monitor during system operation and use during incident analysis to expedite discovery of incident root causes.

IV. Taxonomy of Cloud Monitor Data

To support coordinated analysis of heterogeneous monitor data, we define a general taxonomy of the types of fields present in the data. The taxonomy has two purposes: 1) to support labeling of all fields present in metric and log data so both may be handled uniformly, and 2) to facilitate automated extraction of time series features.

We identify the following field types across all data sources:

1) **Timestamp**: Fields representing the time at which a metric was sampled or a logged event occurred.
2) **Identity**: Fields that uniquely identify a metric measurement or log entry. For metric data, the monitor type and server name are often the only identity features required, but when multidimensional metrics are sampled conjunctly, additional identity features may be required. For example, for the Linux top utility, process ID must be used as an identity field to uniquely identify the per-process memory utilization metric.

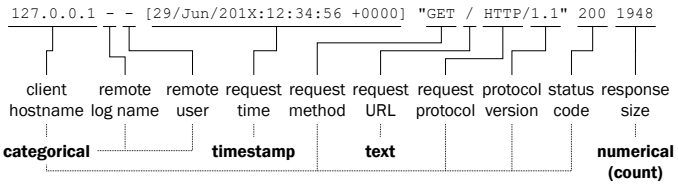

Fig. 1: Field type labels for all fields in Apache access logs. The name of each field is given below the sample log line, and the field types are shown in bold below the field names.

3) **Metadata**: Fields with values that do not change over time for a given feature or data source. Typically, metadata describe some property of the generating process (e.g., the hostname for syslog messages). Metadata fields can be recorded exactly once for a particular feature.
4) **Numerical**: Most fields from metric data sources are either numerical or categorical (see below). Numerical fields describe quantitative measurements.
5) **Categorical**: Fields that describe qualitative measurements with a discrete, often small range of possible values. Examples include error code and HTTP method.
6) **Text**: Fields that contain semi-structured or unstructured messages of arbitrary length or value, such as those coded into software (e.g., logging statements), generated by users (e.g., request URL strings), or generated randomly.

Numerical fields can be further classified into one of two types: cumulative or instantaneous. Cumulative fields capture the cumulative behavior of the underlying process over a given sampling time interval. Examples include CPU utilization, network traffic volume, and request counts. Within cumulative fields, there are two subtypes that depend on whether the values are normalized (averaged) with respect to the sampling interval. If they are, they represent the *rate* (or proportion) of activity. If they are not, they represent the *count* (or quantity) of activity. Instantaneous fields measure the *state* of the underlying process at the time of sampling. Examples include current memory utilization and running process count.

To illustrate the use of our taxonomy, consider the field labels for the Apache access logs shown in Figure 1. Aside from the obvious timestamp field, we would classify the response size field as a numerical feature describing a count (of bytes sent) and the request URL as a text field, as it can contain arbitrary data. We would classify all other features as categorical.

V. Automated Feature Extraction and Reduction

A. Feature Extraction

Our monitor field taxonomy enables us to automate feature extraction and discovery by defining steps to be taken when processing fields of each type.

Timestamp fields are used directly as the timestamps for extracted features. **Identity** fields are used to separate monitor data into logically distinct *streams*, each of which can be thought of as a separate monitor. **Metadata** fields must be unique per identity field, so they serve to validate the uniqueness requirement of the identity fields.

The next two field types describe observations and measurements made by the monitors, so they are used to generate the values of the extracted features. The values of **numerical** fields are used directly as the values of extracted features. **Categorical** fields, on the other hand, are similar to identity fields in that they describe classes of activity in the system. We therefore treat them much like identity fields.

Finally, **text** fields constitute the unstructured component of log messages. Text fields often contain information very pertinent to incident analysis, but extracting the information can be difficult because of format issues. For example, in the case of the WAS logs in Listing 1, admins were interested in distinguishing errors by their application exception type, but this information was buried in the text component of the logs, which made robust feature extraction difficult.

Rather than reinvent the wheel, we leverage existing techniques in unsupervised log parsing [7] [10] to process text fields. These techniques identify log message formats and parameters by distinguishing between strings that are constant across all messages of a particular format (labeled as *log keys*), and those that are variable (labeled as *parameters*). We note that a recent survey [7] shows that for most log types, it is possible to achieve high parsing accuracy with limited preprocessing and tuning, which we found to be the case, as we describe below.

After the message formats and parameters have been extracted, our framework allows administrators to label relevant parameters for message formats of interest with the field types from our taxonomy. Once the parameters have been labeled, we handle them as if they were structured fields, aggregating, separating, or ignoring them accordingly. That is one respect in which our approach improves upon previous work. We treat the message formats as categorical fields.

We claim that manual labeling of relevant parameters is tractable for most cloud log sources because the number of unique message formats for a given cloud system is relatively limited and changes slowly; in our dataset, the three monitors for which we processed text fields contained 89, 32, and 79 unique message formats. Where the number of log formats is too large, we believe that labeling of a small number of initial parameters as categorical features can still provide valuable feature separation, as early positional parameters often contain message codes that fully specify the remaining message format.

In our implementation, we used a modified version of the longest common subsequence (LCS) based parsing approach proposed in Spell [10]. Because the text fields in our log messages could take values with multiple lines (separated by newline characters) and could have a large number of tokens (e.g., exception stack traces), to improve parsing efficiency, we chose to examine the longest common prefix instead of the LCS, and bounded the number of tokens considered. We also preprocessed our logs by replacing tokens that matched some common parameter types (e.g., IP addresses, numbers, Java class names, dates) with placeholders, as suggested in [7].

Figure 2 shows a sample of the hierarchy of message formats we obtained for the WAS error logs. As we were primarily interested in the `app error code` and `Java exception class` parameters, we labeled those parameter values as categorical fields within our taxonomy and ignored all others. In general, however, a user of our framework could label all parameters and rely on the feature reduction stage we describe in Section V-B to remove redundant and non-descriptive

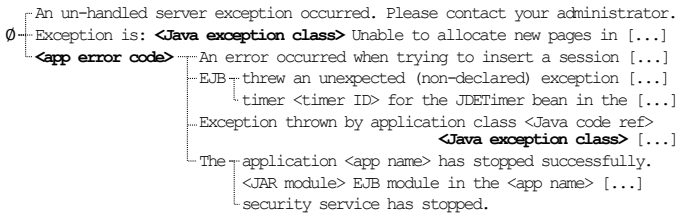

Fig. 2: Example of the hierarchy of message formats for the unstructured text field in the WAS error logs. Parameters are denoted by parameter descriptions enclosed in angle braces.

features; we discuss this further in Section VII.

Procedure for automated feature extraction: Our framework performs feature extraction in two stages. First, we process data from each monitor individually to extract a set of time series, which we refer to as *features*. Second, to allow features from different monitors to be analyzed uniformly, we *summarize* features from all monitors by resampling them to a common sampling interval and aggregating them based on the field type.

Stage 1: Extraction

The procedure to extract features for a monitor is as follows.

1) Convert all text fields to categorical log keys and parameters as described above and drop the text fields.
2) Split the data into separate *streams*, grouping by identity field values.
   a) Each metadata field should take the same value per stream. Verify that this is true and save the metadata field value for each stream.
   b) Each numerical or categorical field now defines a *proto-feature* keyed on the identity field values, with *samples* for each timestamp value.
3) Identify the *feature ID field sets* for which features should be extracted. A feature ID field set is a set of categorical fields (including those derived from text fields) that, together with the identity field(s), will uniquely identify a feature. By default, a singleton feature ID field set is created for each categorical field. We also used domain knowledge to define additional ID field sets for some monitors.
4) For each feature ID field set, split the proto-features into *features* by grouping samples that have the same combination of values for each field in the feature ID field set. Features that correspond to numerical proto-features take the value of the proto-feature; those that correspond to categorical ones take a value of 1 for each sample.

To demonstrate, recall the fields for Apache access logs from Figure 1. The fields we extracted for the EPC web server were request time, response size, client hostname, request method, protocol version, and status code; values for the other fields were empty or constant, so we omitted them. We created proto-features for response size (the only numerical field) and request count (taking a value of 1 for all samples). The last four fields are categorical, so we created singleton feature ID field sets for each. Based on our domain knowledge, we also chose to create feature ID field sets for {status code, request method} and {client hostname, status code, request method}. Feature extraction using our framework yielded 98 distinct features for incident 1's Apache access logs.

In Table II, we show the total number of features extracted for each incident in our case study dataset. Our feature extraction approach intentionally generates a large number of features, as our objective is to extract as many meaningful features from the data as possible; in Section V-B, we show how we reduce the features to a manageable number.

Stage 2: Summarization

The features we obtain from feature extraction will naturally have varied timestamps, and, in the case of metric data, may also have varied sampling intervals. In order to facilitate cross-monitor feature analysis, we subsequently *summarize* the features by resampling all features. That is, we bucket data points for each feature into contiguous time windows of fixed size (the resampling interval) and aggregate their values to produce the summarized features.

In general, the resampling interval should be chosen based on domain knowledge of the dynamics of the system in question—smaller values allow for finer granularity in feature comparison, whereas larger values support meaningful analysis of behaviors that manifest over long timescales. It may also be valuable to try multiple resampling intervals for different types of analysis. Based on our knowledge of our system, in our experiments, we chose a resampling interval of 10 seconds.

The appropriate aggregation method for each feature is defined by our taxonomy: features representing rates are *averaged*; those representing counts and categorical proto-features are *summed*; and those representing states have a representative value chosen (e.g., maximum or most recent).

B. Feature Clustering

To promote effective analysis of the data, and to remove redundant and uninformative features, we propose a hierarchical feature clustering method that groups highly correlated features across all monitors. We now describe the method, and illustrate its effectiveness using the EPC dataset.

Feature clustering is useful for two reasons. First, many features are likely redundant. Within each monitor, many fields represent similar values, and in some cases, splitting on a categorical field does not yield a meaningfully different feature. Furthermore, some monitors collect redundant data (e.g., some OS-level vs. per-process performance metrics). Clustering can reduce the number of such features an admin must examine. Second, as we illustrate in Section VI, clustering can uncover relationships between features across different monitors that are important in incident detection and root cause analysis.

Before clustering the features, we first eliminate uninformative features. Some features extracted by our framework will either have no value over the entire dataset (*empty* features), or have a constant value (*constant* features). For example, we found that some combinations of status code and request method did not appear within the EPC's web access logs, but our framework created a feature for them. Empty and constant features made up roughly 15% of all features in our dataset.

Next, we observe that many features derived from logs are sparse (i.e., rare events)—they are zero-valued the vast majority of the time—and should be clustered separately from dense features. We define a feature $f(t)$ to be *sparse* if $\sum_{t_i \in [t, t+w)} [f(t_i) \neq 0] < R$ for some t, where R is the rare event occurrence threshold and w is the rare event window, and

TABLE II: No. of features per incident at each stage of reduction

Feat. reduction stage	Incident 1		Incident 2		Incident 3		Total	
	Dense	Sparse	Dense	Sparse	Dense	Sparse	Dense	Sparse
Original features	3693		2132		2231		8056	
Without empty feat.	3327		1495		1704		6526	
Pre-clustering	2238	1089	484	1011	426	1278	3148	3378
After clustering (1)	1302	347	357	340	306	625	1965	1965
After clustering (2)	1232	320	325	314	287	494	1844	1128
After clustering (3)	1108	273	296	287	260	475	1664	1035
Reduction %	50%	75%	39%	72%	39%	63%	47%	69%
Num. total clusters	1381		583		735		2699	
Overall reduction %	63%		73%		67%		66%	

dense otherwise. R and w should be defined based on domain knowledge; in our dataset, we chose $R = 3$ and $w = 500$s to separate dense and sparse features. We show the feature clustering results for each incident separately in Table II.

We now propose a hierarchical feature clustering method that groups highly correlated features at three levels: (1) within each unique feature ID field set, (2) within each monitor, and (3) across all monitors. At each level, we compute the pairwise Spearman rank correlation between all features within the level and identify maximal cliques with pairwise correlations that exceed a *feature similarity threshold*. Each clique represents a set of *redundant* features. For each clique, we choose the feature with the highest sum of pairwise intra-clique correlations as the clique's representative feature to be used in the next level of clustering. Hierarchical clustering improves scalability by drastically reducing the number of features at lower levels, where they are more likely to be redundant.

We chose to use Spearman rank correlation because unlike Pearson correlation, as used in [11], which can only identify linearly correlated features, Spearman correlation accurately groups together features that trend together nonlinearly (such as request count and request size). We propose the use of a high feature similarity threshold (i.e., above 0.9) so as to limit the number of spurious groupings; based on examination of our dataset, we chose the value 0.96.

For example, for our EPC Apache access logs for incident 1, our framework first found 11 of the 98 original features to be empty, 68 to be dense, and 19 to be sparse. After clustering was performed, the original features were grouped into 15 dense clusters and 1 sparse cluster. Closer examination of the clusters revealed that all but two of them grouped together requests from hostnames that corresponded to different test roles, and the two anomalous clusters grouped various hosts with error status codes that corresponded to the failure incident.

Table II shows the results of feature clustering for the EPC data. Overall, clustering reduced the number of features by 66%.

VI. Application to Incident Response

We now illustrate some of the ways our framework can improve the effectiveness of enterprise cloud administrators in reliability and performance incident data analysis.

A. Feature Discovery

Our approach enables administrators to discover meaningful features more easily across their heterogeneous monitors, reducing the time and domain expertise required in deciding what features are relevant for analysis.

As we describe in Section III, prior to the development of our approach, administrators of the EPC needed to decide a priori which logs and metrics they considered useful for incident analysis, and needed to construct dashboards manually for each. Because of the engineering cost of developing tools to perform complex analyses of the log data, the administrators often used rudimentary filters, such as message severity level, token count, and keyword presence, to define metrics based on the logs. In some cases, where documentation stated that fields from the structured part of the log described the unstructured messages, they ignored the unstructured messages outright.

In our examination of the EPC dataset, we found that those simplifying assumptions overlooked many features salient to the incidents that took place.

Identifying unique feature ID field sets: The documentation for the WSAM error log states that the message number field uniquely identifies the unstructured message format. As a result, the EPC admins used the message number field to count error messages of different types. However, our analysis revealed that message number was common to multiple error message formats, and that the set of (source line, message number) was required to uniquely distinguish error types.

Feature separation by parameter value: Even within errors of the same type, meaningful signals were sometimes hidden within parameters of the unstructured message. For example, for incident 3, EPC admins found that during periods of service unavailability, there was a spike in a WSAM error message that corresponded to loss of network connection with another server, but as the IP address of the unresponsive server was contained in the unstructured message, the admins were forced to manually investigate the logs each time the error message spiked to identify the server. Our framework enabled us to automatically separate the features by server IP address; the resulting subfeature for the WAS IP address clustered with an error that corresponded to an SSL version issue, which had been identified as the root cause of the incident.

B. Feature Deduplication

Another major challenge the administrators faced was identification of meaningfully unique features within the data. While it was obvious that OS performance metrics collected from the nmon and top monitors provided duplicate information, it was not clear whether, for example, failed requests in the web access logs were similarly correlated with WAS errors. The admins therefore made assumptions based on past experience which of the thousands of candidate features to monitor.

Our framework automatically groups together strongly correlated features, allowing redundant features to be monitored once. We found many instances in the EPC where the use of our framework would have prevented the creation of dashboards for multiple log message types that were ultimately redundant.

C. Root Cause Analysis

Finally, an understanding of which features are strongly correlated can help admins uncover insights about their system that can aid in root cause analysis. For example, in incident 2, the database server ran out of free disk space, causing a spike in WAS errors for application requests that required database writes. As a consequence, the web server started

responding to requests with a status code of 500 (Server Error). When we used our framework to examine the features clustered together with the feature for status code = 500, we found that it strongly correlated with a feature for a specific WAS error type. Examination of the clusters that corresponded to the subfeatures for that error type revealed a large cluster of 12 features, one of which was a SQLException with the message "Unable to allocate new pages in table space", which indicated that the root cause of the incident was likely a disk space issue. While explicit monitoring of disk utilization would have identified it as a possible cause, our approach enabled us to trace the errors back to the disk space issue directly.

VII. Discussion

Generality: Our approach can generalize well to other cloud systems and log types. We have demonstrated its use for 8 considerably different log types in the EPC; practitioners interested in applying the approach can follow the procedure we have laid out to the monitors in their own system. In other, ongoing work, we have successfully applied our framework to syslogs, firewall logs, and security event logs.

Scalability: The most expensive operation in our framework is the clustering-based feature reduction. While the cross-correlation step scales quadratically in the number of features and linearly in the length of the data, we found that because of high redundancy at levels 1 and 2 of the clustering, the number of features clustered across all monitors (level 3) is manageable. Our Python implementation utilizes the Pandas library [12] to compute correlation and the Bron-Kerbosch algorithm to find cliques. It was able to process all features in our dataset (which contained data from over 2 weeks of system operation) in a few hours while running on a single server with an 8-core Intel Xeon CPU E5-2450 processor and 64 GB of memory.

Extensions to feature extraction and clustering: We have identified many alternative ways to extract features, and we intend to examine them in future work. They include joining of features across different monitors (e.g., on ID fields as in [13]), and parsing of fields in alternative ways, such as by counting the number of unique values taken by categorical features or by computing interarrival times for sparse features.

VIII. Related Work

The problem of extracting meaningful signals from heterogeneous log data for failure detection has been studied extensively in the high-performance computing (HPC) literature. LogDiver [11], Desh [14], and LogAider [15] all extract features from HPC datasets, including hardware error, reliability, workload, and scheduler logs, using extensive domain knowledge of the log semantics and structure. They then cross-correlate features spatially and temporally. While those solutions are useful, they are highly specialized for HPC problems, and do not provide a method to extract features from arbitrary monitors.

Among commercial tools for cloud log analysis, VMWare vRealize Log Insight [16] and Microsoft Azure Kusto [4] provide the most similar functionality to our framework, but our work is still distinct. Kusto does not support extraction of fields from unstructured log messages, and while Log Insight does, it does not allow parameters to be treated like structured fields. Furthermore, while both support clustering of features—Kusto by common discrete field values and Log Insight by message format—clustering is limited to features within individual monitors, whereas we perform cross-monitor clustering.

Our work relies on unsupervised approaches for identifying message formats and parameters within unstructured log data. Zhu et al. [7] examine 13 mainstream approaches that utilize various techniques (data mining, clustering, heuristics, etc.). They find that many cloud log types can be accurately parsed with existing approaches. We extend the utility of those techniques by enabling the extraction of meaningful time series features from parsed logs in conjunction with metric data.

IX. Conclusions

Existing tools and techniques for parsing cloud monitor data cannot handle diverse data types without considerable manual administrator effort. We propose a monitor data analysis framework for enterprise clouds that allows administrators to semi-automatically process heterogeneous data from multiple levels of a cloud platform into a manageable set of meaningful time series features that are useful in incident analysis. We illustrate the use of our framework on data from an experimental PaaS cloud and demonstrate its utility in reducing the complexity of monitor data analysis for incident response. In future work, we intend to study how the features we have extracted can best be used to detect anomalies and support rapid incident response.

Acknowledgments

We thank Jenny Applequist for her editorial assistance, and Yu Gu, Richard Harper, and Mahesh Viswanathan for many constructive discussions and feedback. This work was supported in part by an IBM Ph.D. Fellowship.

References

[1] W. Xu, "System Problem Detection by Mining Console Logs," Ph.D. Dissertation, University of California at Berkeley, Aug. 2010.
[2] C. Sridharan, "Logs and Metrics," Medium, Apr. 2017. [Online]. Available: https://medium.com/@copyconstruct/logs-and-metrics-6d34d3026e38
[3] D. Reichert, "Logs and Metrics: What are they, and how do they help me?" Sumo Logic, Jan. 2018. [Online]. Available: https://www.sumologic.com/blog/logs-metrics-overview/
[4] "Reference - Azure Data Explorer," Microsoft, Accessed: 2019-08-31. [Online]. Available: https://docs.microsoft.com/en-us/azure/kusto/
[5] "Enterprise Log Management for Any Scale," GrayLog, Accessed: 2019-08-31. [Online]. Available: https://www.graylog.org/products/enterprise
[6] "Machine Data Management & Analytics," Splunk, Accessed: 2019-08-31. [Online]. Available: https://www.splunk.com/en_us/software/splunk-enterprise.html
[7] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and Benchmarks for Automated Log Parsing," in *Proc. 2019 41st ACM/IEEE Int. Conf. Softw. Eng.*
[8] Graphite, Accessed: 2019-08-31. [Online]. Available: http://graphiteapp.org
[9] "ELK Stack: Elasticsearch, Logstash, Kibana," Elastic, Accessed: 2019-08-31. [Online]. Available: https://www.elastic.co/elk-stack
[10] M. Du and F. Li, "Spell: Streaming Parsing of System Event Logs," in *Proc. 2016 IEEE 16th Int. Conf. Data Mining*, pp. 859–864.
[11] C. D. Martino, S. Jha, W. Kramer, Z. Kalbarczyk, and R. K. Iyer, "LogDiver: A Tool for Measuring Resilience of Extreme-Scale Systems and Applications," in *Proc. 2015 5th Workshop on Fault Tolerance for HPC at eXtreme Scale*, pp. 11–18.
[12] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proc. 9th Python in Sci. Conf.*, 2010, pp. 51–56.
[13] A. Bohara, U. Thakore, and W. H. Sanders, "Intrusion Detection in Enterprise Systems by Combining and Clustering Diverse Monitor Data," in *Proc. 2016 ACM Symp. and Bootcamp Sci. of Secur.*, pp. 7–16.
[14] A. Das, F. Mueller, C. Siegel, and A. Vishnu, "Desh: Deep Learning for System Health Prediction of Lead Times to Failure in HPC," in *Proc. 2018 27th ACM Int. Symp. High-Perform. Parallel and Distrib. Comput.*, pp. 40–51.
[15] S. Di, R. Gupta, M. Snir, E. Pershey, and F. Cappello, "LOGAIDER: A Tool for Mining Potential Correlations of HPC Log Events," in *Proc. 2017 17th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Comput.*, pp. 442–451.
[16] "Log Management Tool And Analytics — vRealize Log Insight," VMWare, Accessed: 2019-08-31. [Online]. Available: https://www.vmware.com/products/vrealize-log-insight.html