Chapter 3

Metric Selection and Unsupervised Anomaly Detection Using Log-Metric Regression Analysis

This chapter provides a conceptual framework and presents a structured view to the approach that is proposed in this thesis. First, overview of the steps of the research approach is outlined then the mechanisms that are employed on each of these steps are explained and elaborated.

3.1 Overview of the Proposed Approach

The proposed approach in this thesis attempts to address the challenges and limitations outlined in Section 1.1 and to fulfil the objective of the thesis that highlighted in Section 1.2. In particular, the approach proposed in this chapter attempts to propose a metrics selection and anomaly detection technique for system health and performance monitoring. Unlike many anomaly detection methods, the proposed approach is an unsupervised approach, which does not require anomalous labeled data [53, 54]. Also, it proposes a Contextual-based anomaly detection technique. Contextual-based technique (also referred to as Behavioural or Conditional anomaly detection technique) means observed data instances are considered anomalous in association with behavioural attributes of a system [29].

Logs represent the behaviour (context) of system operation while metrics show the health and performance status of a system. Several studies have leveraged logs for failure detection and diagnosis, and a large amount of studies have adopted metrics for system health and performance anomaly detection [32, 29, 14]. However, using the combination of these two valuable sources of information have not been well integrated, perhaps because of the difficulties in mapping these two types of data.

Event logs are textual content while system resource metrics are often represented with numerical values. As can be seen in Fig. 3.1 and Fig. 3.2, event logs are generated at various points in time, and metrics are collected at potentially different points in time. Given Fig. 3.1:

- ***Events*** are referred to individual log lines. We assume each log line has a timestamp and a description.
- ***Clustering*** is a process that clusters fine-grain correlated event logs to set of coarse-grain activities. We referred to this process as log abstraction or log clustering.
- ***Activities*** are referred to set of log events that together are responsible for performing an action. For instance, retrieving user profile information, launching a VM instance, etc.
- ***Metrics*** represents status and utilisation of system resources within a time-window. For instance, average CPU utilisation or network usage within last five seconds.

Detailed system monitoring is often a cost intensive process, and the monitoring metrics are often collected with longer frequency than logs **??**. Moreover, collecting monitoring data usually happens at fixed intervals[1]. In contrast, observation of system operations behavior through event logs happens at non-fixed intervals, such as the occurrences of few event logs within one second, and then the absence of any event logs for the next few seconds or even minutes.For example, Fig. 3.2 shows sample of JBOSS Application Server, by looking at logs timestamps we can observer that logs are happening with various time

[1]such as every five seconds, minute, or every 5 minutes

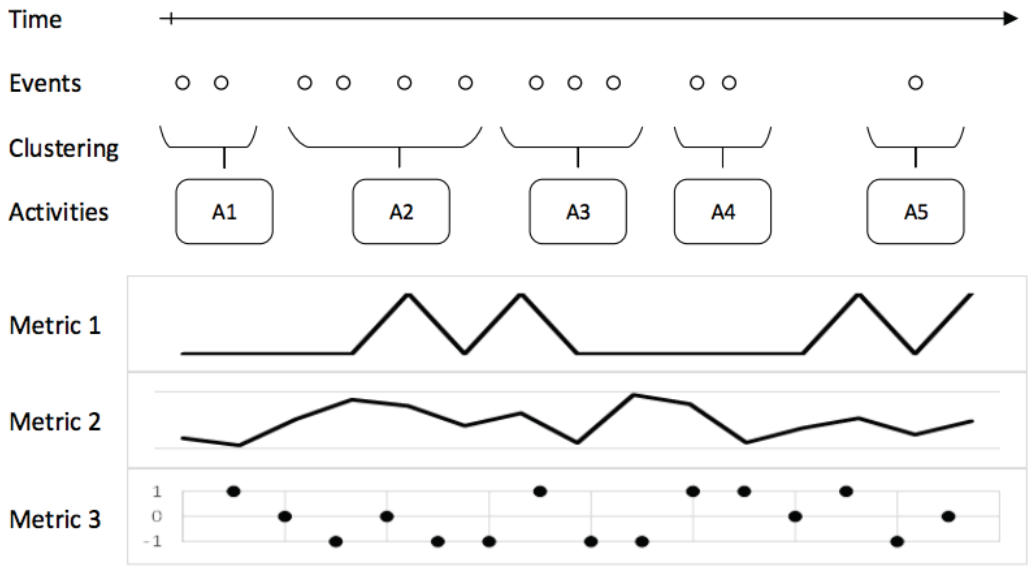


Figure 3.1: Assertion derivation from log and metric observations. *Note: A1..A5 are activities; Metric1..3 are metrics like CPU utilization, network usage, number of instances changes, etc.

interval. Also, in this figure we are showing few examples of grouping set of related log events to a set of activities, we will explain the log abstraction process in Section ?? and Section 3.3.2.

In our investigation, we are interested to find the relationship between operation activities derived from log events to the symptoms observed in resource metrics. Once we map logs and metrics, we adopt a statistically justified technique using correlation analysis to narrow down the number of metrics to the most sensitive metrics influenced by application operation behaviour that is reflected in event logs. Then, our approach uses a statistical technique to extract a regression-based model that explains the correlation and potential causalities between operation event logs and resource metrics. To derive assertions from observations, we assume that there is a stream of time-stamped events, such as events represented by log lines, and at least one metric that can be observed. The output of the regression model is used to generate assertions, which are then leveraged for anomaly detection of run-time execution of application operations.

Figure 3.2: An example of JBOSS logs with clustering related log events to set of activities.

In our investigation, we are interested to find the relationship between operation activities derived from log events to the symptoms observed in resource metrics. Once we map logs and metrics, we adopt a statistically justified technique using correlation analysis to narrow down the number of metrics to the most sensitive metrics influenced by application operation behaviour that is reflected in event logs. Then, our approach uses a statistical technique to extract a regression-based model that explains the correlation and potential causalities between operation event logs and resource metrics.

To derive assertions from observations, we assume that there is a stream of time-stamped events, such as events represented by log lines, and at least one metric that can be observed. The output of the regression model is used to generate assertions, which are then leveraged for anomaly detection of run-time execution of application operations.

The high-level steps of the approach, also shown in Fig. 3.3, are as follows:

- 1) data collection and data preparation;
- 2) metrics derivation;
- 3) Log abstraction and log-metrics data mapping;
- 4) correlation derivation between logs and metrics;
- 5) metric selection;
- 6) and assertion specification for anomaly detection.

We describe the steps of data collection for both monitoring metrics and logs in this section, whereas the next section captures the steps for utilising these data for metric selection and anomaly detection, respectively.

3.2 Sources of Monitoring Data and Data Preparation

3.2.1 Event Logs from Operation Tools

While the status of resources is often monitored through metrics, logs are the main source of information that reports what functions are executed in a sys-

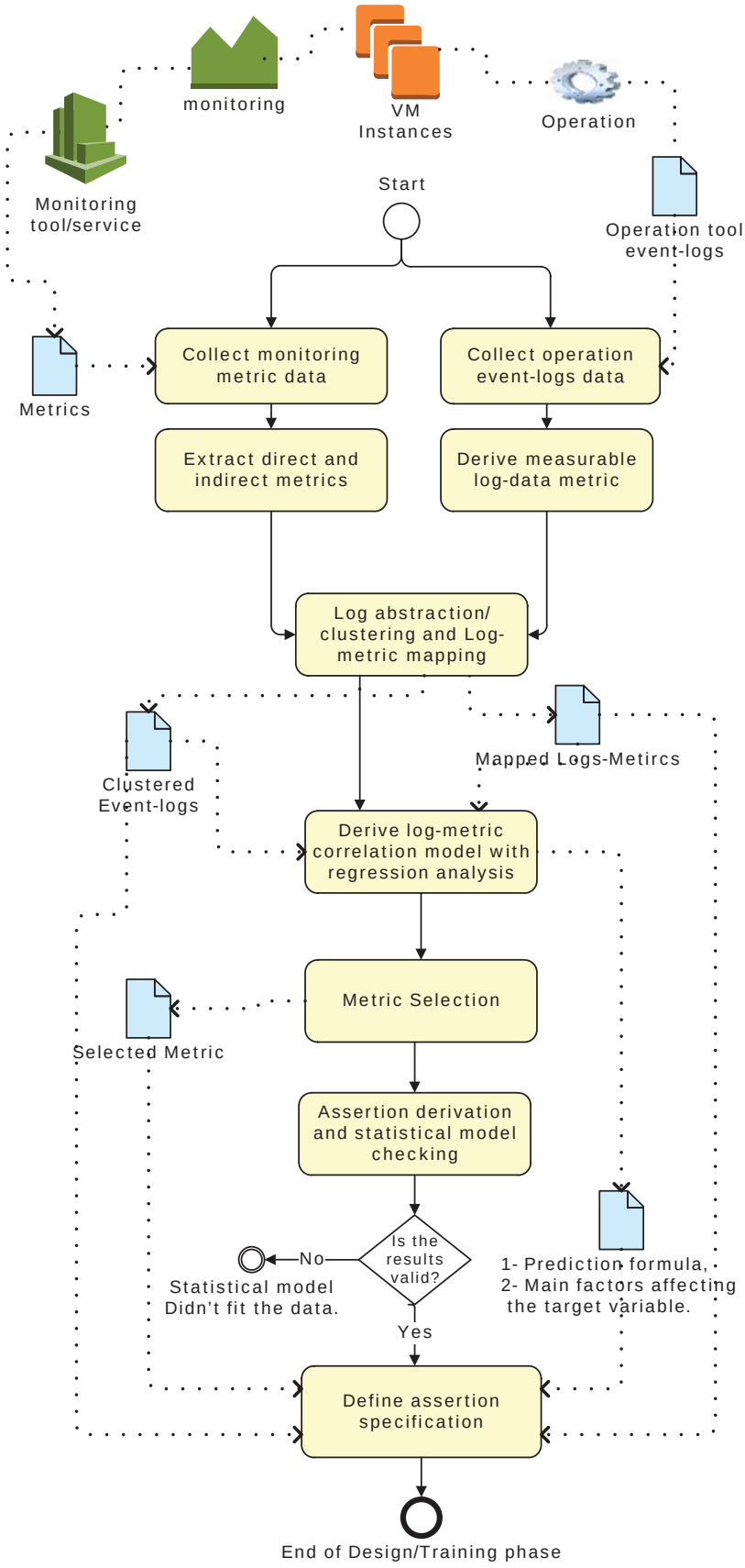


Figure 3.3: Workflow of the proposed framework.

tem [55]. Log events explain the behaviour of the applications and operations and in our contextual-based anomaly detection approach is the main source to extract the events of a system [55, 27].

In the approach presented in this thesis, to draw mappings between numerical metrics and contextual logs, we needed to extract a set of metrics that show the occurrences of different event logs. Although the styles of logging might be different, almost all type of logs contains time-stamped information, whether they are application logs, database logs, or operation logs. Furthermore, log message represents information about an event, including logs that indicate preparation or waiting periods.

The timestamp and description attributes can be seen in almost all operational logs, though the quality and granularity of logs may differ from one application to another. For instance, middle ware application servers and operational tools such as Apache Tomcat[2], Eclipse Jetty[3],JBOSS [4], Asgard[5], and CloudTrail[6] provide logs with above formats(logs with timestamp and event description) . Hence, this study assumes a logged event must have at least two attributes: a *timestamp* and an *event description*. Knowledge derived from logs can be leveraged to monitor the behaviour of a system and detect errors and diagnosis the root cause failure [26, 56]. In the approach proposed in this thesis, we use the information from logs along with a status of metrics to identify anomalies in a system operation.

3.2.2 Metrics from Monitoring Tools

The approach of this study requires two sources of monitoring data: one is the log events of running operations and the other one is the metrics from resources. Both of these types of monitoring data are available in almost all enterprise application systems. For examples, public and private cloud service providers offer monitoring services namely: Azure Monitor for Microsoft Azure[7], Google

[2]https://tomcat.apache.org/
[3]http://www.eclipse.org/jetty/
[4]http://www.jboss.org/technology/
[5]Netflix Deployment Tool - https://github.com/Netflix/asgard/wiki
[6]Logs reporting service of Amazon AWS - https://aws.amazon.com/cloudtrail/
[7]http://azure.microsoft.com/

Stackdriver for Google cloud[8], Cloudwatch for Amazon AWS[9], LogicMonitor [10] and many other third-party services and application performance monitoring solution providers[11].

Monitoring tools provide multiple metrics to show the status of various system and application resources. For example, at the operating system level, one can collect monitoring data of running processes, system activities and hardware and system information either using native Linux tools like "top" and "vmstat" or utilising more sophisticated monitoring solutions like Nagios[12]. As another example, public cloud service providers such as Microsoft Azure and Google Cloud provide several monitoring metrics which are available at both individual resource level (e.g storage, or VM instances level) and at a group level showing the aggregated form of multiple resources. (e.g. the average disk write in 10 storage devices).

Typically, system monitoring can be conducted in two forms: one through performance-based metrics, and the other by state-based metrics.

- **Performance or direct metrics**: These basic metrics are often available by default with the monitoring tools and available for most of the metrics that show the capacity or performance of various resources. This type of metrics are the metrics that show the percentage or the degree of the changes on the usages of the resources, such as CPU utilization, disk read IO, memory usage, network traffic, number of SQL thread running, number of open connections, etc. - list of hundreds of these metrics from Google cloud platform metrics, Stackdrive Monitoring Agent metrics and Amazon Web Services metrics can be found in the link in the footnote [13].

- **State-based or indirect metrics**: These metrics may not always be available directly and they often indicate of transition of status of re-

[8] http://cloud.google.com/stackdriver
[9] http://aws.amazon.com/cloudwatch/
[10] http://www.logicmonitor.com/
[11] http://haydenjames.io/20-top-server-monitoring-application-performance-monitoring-apm-solutions/
[12] https://www.nagios.com/
[13] https://cloud.google.com/monitoring/api/metrics

sources from one state to another. One important example of this type of metrics is the ones indicating the status of a VM. For example Cloud-Watch (AWS monitoring service), includes no metric that explicitly show the number of instances started or terminated within an auto scaling group. Fig. 3.4 shows a typical state flow of the lifestyle of a VM [57].

In Fig. 3.4:

- **Uninstantiated** is the pre-creation or baseline state of a VM and my involve platform-dependent operations. For instance, an AMI and the configuration of the VM that VM will be created based on that.
- **Waiting** often refers to the state of resource allocation and booting process.
- **Running** is the state of a VM that is ready to provide services.
- **Paused** refers to a temporary suspension of VM which is stored in the memory.
- **Suspended** is also called "stopped" indicates a suspension of a VM but the state of the VM is persisted on the physical disk space rather than the memory.
- **Terminated** is the state which a VM is destroyed and thus can not be restored.

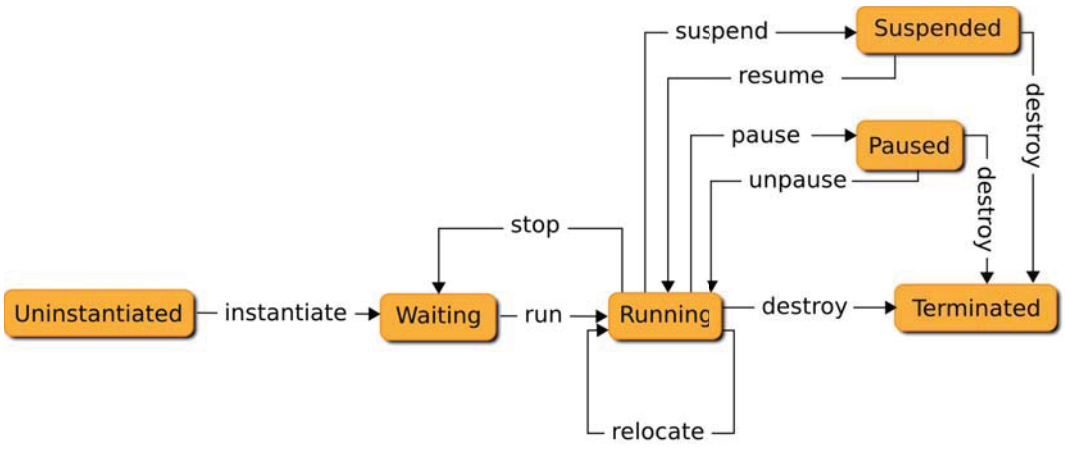


Figure 3.4: Sample of a lifecylce of a virtual machine with transient actions from one state to another.

In this thesis, we will explore both types of these metrics and evaluate the applicability and effectiveness of our approach for each of these groups.

As there is usually a large number of monitoring metrics available, a system operator needs to decide which monitoring metric or metrics to focus on. In Section 3.4.1, we will demonstrate how to leverage a statistical supported technique to identify most sensitive metrics for detection of anomalies in a system.

3.3 Log Processing and Log Abstraction

3.3.1 Representing Logs as Quantitative Metrics

Raw log event are often unstructured, as developers write a log message using free text for convenience and flexibility [15]. Also, system logs are often available in a large number and they are fine-granular [27, 58]. This makes the monitoring and troubleshooting of systems using raw logs to be a cumbersome task, especially in large-scale computing environments. For instance, the number of registered log events in an application server such as Tomcat or JBOSS can reach over tens or hundreds of log lines per minute. Nevertheless, most of the log events are often recurring, indicating of occurrences of similar events.

For our study, to observe how the activities reported on log events change the state of resources we are interested on tracking reoccurring log events. So, we needed a way to parse and trace log events. A typical approach on log parsing is to utilize regular expression to derive a template of event logs [35, 30, 59, 60, 61]. Similarly, we employed the common approach of adopting regular expression to identify log event types. In this process, the raw log messages are transformed to a set of structured log events.

As shown in the example in Fig. 3.2, log messages typically come with a timestamps and a message content. Log message describes what is happening at runtime [61]. Thus, for the log processing in our approach, we assume logs contain both timestamps and message description. The message part of a log, can be divided into two parts: the constant part and the variable part [35]. The constant part of the message is fixed and remains the same for every log.

The variable part holds the runtime information of log events like a an IP address, a port number, and etc [35].

The goal of log parsing in our approach is to extract the pattern of recurring event logs by automatically separating the constant parts and variable parts of a raw log message, and further transform each log message into a specific log event type. To fulfil this aim, we process log files through the following steps. First, the timestamps and log description are extracted for each log line. Followed by tokenization of the log message. Next, regular expression are generated for each token of the log message. Then, message tokens are divided to two parts of constant tokens and variable tokens by analysing the pattern of regular expression using a set of pre-defined rules. Lastly, the generated regular expression is combined form a unique log event type. For each new log line, the log event is compared by pattern matching with regular expression and if the pattern is not found, the above steps will be repeated for the new log line.

The rules we used in our log analysis with regular expressions includes the common pattern of url address, IP address, port number, camel-case strings, and numerical values (which may include time duration, IDs, etc.). This method is simple and shown to be effective in past studies [59, 19, 25], however, these rules may need to be configured or extended for parsing logs of a new project.

In the process of log transformation, the variable tokens are replaced with the regular expression for the token type. For instance, string token *process[11043]* is replaced by *process[\d+]*, "\d+" indicates that the string can have numbers as a variable in the brackets; the rest of the token taken as exact expression. Another example log transformation is shown below:

```
"Remove instances [i-08b43bc7] from Load Balancer ELB-01"
```

generate the following regular expression:

```
"Remove\sinstances\s\[i-[a-z0-9]+\]\sfrom\sLoad\sBalancer\s.*"
```

The output of the the above step is a set of regular expression patterns that represent unique types of event logs.

Once we parsed the logs and identify what type each log belongs to, we can start counting the occurrence of each log event type, however, we need a mechanism to map the counting extracted from the event logs to the time-window that our system health and performance metrics are available. For instance, if metrics are available with one-minute intervals, then to model the relationship between logs and mcometrics we need to represent metric extracted from the logs with the same time-window.

To the best of our knowledge, this matter has not yet been explored in previous approaches for log analysis perhaps as most of the past approaches had the sole focus on logs, rather than considering other factors like metrics. Also, unlike many past studies [62, 63, 61, 58, 34] that employ feature extractions in order to track what happens in a logs, in our approach, we propose a simple method that does not have interest on interpreting the context of the log message.

In fact, the focus of our thesis is to find out whether the mere occurrence of individual log events, without having domain knowledge of the context of the logs, can be employed along resource metrics in order to improve metric selection and anomaly detection. In this direction, and since in many system resource monitoring, such as in case of public cloud services, there are limited options available on configuring the frequency of resource monitoring, we propose a weigh-timing method that performs counting based on *interpolated occurrence strength* of each log event within a time-window.

In this method, the *interpolation* indicates at which unit (e.g. second) of a time-window(e.g. minute) an event happened. Indicating a point of time for the derived metric for log events would show a relative interval of the happening of a set of log events. To this end, we parse the timestamp of each log message and extract the point of time (e.g. seconds of a minute) the event happened. Then a relative occurrence value is calculated as an interpolated value, capturing the time-wise proximity of the event to the full time-window (e.g. minute) a time-window before and after the event happened.

Given a log event type denotes as e, the smallest unit of time that logs can be tracked to be denoted as x, the interval of time period that monitoring

metrics are available to be denoted as tw, and D_{tw} represents the duration of time window, then the weight-timing occurrence of an event type at time x of a full time-window can be obtained:

$$e_{n(tw)} = \frac{D_{tw} - x}{D_{tw}}$$

The weight-timing for the interpolated value of the above event for the next minute can be obtain from:

$$e_{n(tw+1)} = \frac{x}{D_{tw}}$$

So for the sum of n times concurrences of an event type for a time window we have:

$$E_{tw} = \sum_{i=1}^{i=n} e_{n(tw)}$$

The outcome of above step is a matrix of having interpolated occurrence strength for each event type at each time window. This outcome allows us to employ statistical analysis and find correlation between log event types and clustering log events to a set of activities. By this approach, also, we are able to map the event logs into the time-window interval of the monitoring resource metrics of our system.

Table 3.1: Coefficient correlation output of OLS regression

Timestamp	E1	E2	....	Ek
$tw1$	$E1_{tw1}$	$E2_{tw1}$	...	Ek_{tw1}
$tw2$	$E1_{tw2}$	$E2_{tw2}$	...	Ek_{tw2}
$tw3$	$E1_{tw3}$	$E2_{tw3}$	...	Ek_{tw3}
...	...	...	...	...
...	...	...	...	...
twn	$E1_{twn}$	$E2_{twn}$	...	Ek_{twn}