Proceedings of the International Conference on Inventive Computing and Informatics (ICICI 2017)
IEEE Xplore Compliant - Part Number: CFP17L34-ART, ISBN: 978-1-5386-4031-9

# *Data Driven Decision Making for Application Support*

Karun Thankachan
Software Development Engineer
Dell International Services Private Limited
Bangalore, India, 560047
karun_thankachan@dell.com
karunthankachan1@gmail.com

***Abstract*— This paper discusses the design of a solution to enable data driven decision making for application support. The design proposes a novel standard for logging within applications. The next phase of the design proposes a novel method to use the standardized logs to map the functioning of an application to a finite state automata. The analytics proposed in the design helps understand issues during application processing, assess the impact of issues and quickly take decisions to resolve the issues. Big data analytics and probabilistic models are used on the historical application logs to further predict issues prior to their occurrence, assess the health of application functioning and be able to proactively act on situations that can lead to errors.**

***Keywords*— machine learning, finite state automata, prescriptive systems, big data analytics, probabilistic models***

## I. INTRODUCTION

Application support involves ensuring continued delivery of service of an application [12]. It involves fixing either application or system issues (i.e. issues related to code or infrastructure) .Since these are in the production environment they need immediate attention with minimum time to resolve. Each minute spent on fixing issues would be time a defective product is being used to conduct business. This could mean loss of revenue, customer dissatisfaction, loss of brand image etc. When dealing with such issues in the live environment the first few questions that need to be answered are

- Why did the issue occur? (Code, Infrastructure etc.)
- How do we solve it? (Code Fix/Server Upgrade etc.)

This is often followed by other question such as – How do we ensure this issue does not occur ever again, what is the impact of the issue etc. The lesser the time it takes to get answers to these question the faster business can make decisions and ensure services resume as normal. The solution put across is to build a decision support system utilizing application and system logs to understand the issue and how it came to be. This facilitates data driven decision making, ensuring business can make the most well-informed decision. We propose to do this by standardizing the logs and mapping the working of the application to a finite state machine (FSM). From the FSM the decision support system is able to identify the actions that took place to reach the issue, how it is different from the regular application processing and what steps could be taken so as to fix the issue.

The major contributions of the paper are as follows -

1. A novel standard for application logging and a system to convert logs into a finite state automata.
2. A decision support system to indicate how to resolve issues in application processing.
3. Analytics to assess application health and predict issues based on historical data.

The paper is organized into different sections. Section II discusses the work done in this area and projects ongoing. Section III introduces the design of the system with the help of a motivational example - an E-Commerce Site. Section IV discusses the future work that can be done. Section V is the conclusion.

## II. RELATED WORK

Application support has been a critical function for business to ensure continued delivery of service. Overtime it has taken many forms with the current standard being DevOps. DevOps emerged from an effort by businesses to respond more rapidly to market changes. The new approach was designed to ensure that high-quality updated software gets into the hands of users more quickly. Continuous delivery requires that everyone—from developers to testers to employees in user experience, product and operations—collaborate effectively throughout the delivery process, using multiple feedback loops [16]. Many solutions have sprung up trying to reduce the time for turnaround in application support.

Anodot is one such tool gaining traction in the DevOps space to help proactively monitor and assist in application support. Anomaly detection is such a difficult challenge because of the presence of noise as well as the large number of anomalies. To address these issues, Anodot analyzes anomalies in the patterns of anomalies themselves. [2] These algorithms for second-level learning about anomalies can distinguish among more and less significant anomalies. For example, instead of simply looking at anomalies in patterns of CPU usage, Anodot uncovers anomalies in the average CPU usage across a server cluster. Anodot's patented innovation is how it performs real-time machine learning at scale. This combination of real-time anomaly detection across large data sets differentiates itself from other similar products in the market [3].

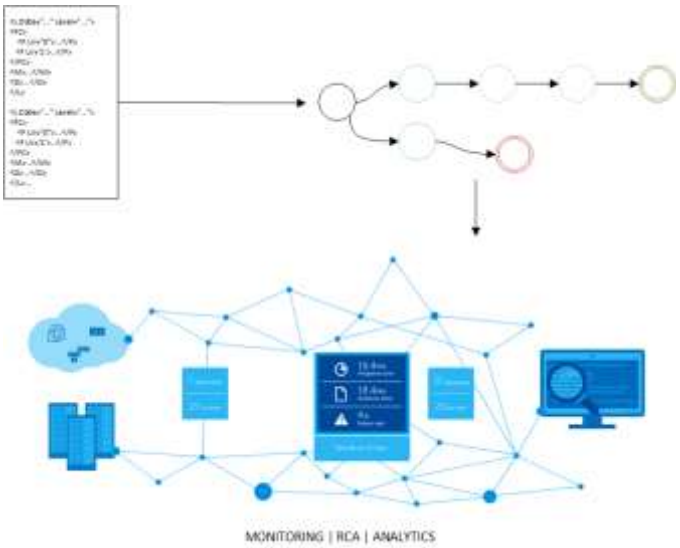
Elastic-Search is yet another tool in the market for analyzing application logs and building interactive dashboards. The recent collaboration with PreLert has placed them as the top

solution when it comes to this space. The underlying technology is called 'Anomaly Detective' [1] [4]. Anomaly Detective uses powerful machine learning analytics to self-learn the behaviors hidden in large data streams. It uses highly accurate statistical analysis to identify the anomalies that are behind performance problems or security threats. Elastic Search also helps to process and clean your logs and format them to produce data as required.

Splunk is yet another tool that has been gaining marketspace. Built as a stand-alone tool with supporting infrastructure to able to collect, index and parse logs from multiple sources, find relationships, monitor and alert on situation it has been a go-to solution for many major companies like Dell, Adobe, Bosch etc. There also exists multiple other solutions such as Numenta [7], Interana [11] and Loom [6] system which work along similar line focusing on anomaly detection within DevOps space [13]. Several other application also help in application monitoring and management such as IBM Tealeaf [14] and AppDynamics [15]. Tealeaf helps to analyze how users interacted with the application and better understand what exactly happened to lead to an erroneous situation in the application. This proves especially helpful from customer-facing applications where the user-interface can be tapped into in multiple ways, all of which the testing may not cover. AppDynamics is another powerful solution that helps to get real-time, end-to-end management made for the most complex and distributed applications. AppDynamics APM helps you focus on what matters with features like application mapping, dynamic baselining, and code-level diagnostics. It is able to track all the interactions that take place between systems and enable you to assess the health of different components in the system. However it cannot delve into an application, break it into logical sections and then assess the interactions between theses logical section which the proposed solution is capable of doing by tapping into the application and system logs.

## III. System Design

The design can be split into three major portions as depicted in Fig.1. The first is a *standard for logging*. The information to be logged includes - the current state the system is in and the changes that are occurring to the system in that state. The second is a method *to search and translate the logs into a finite state machine* (FSM). This filters the logs to a specific date time range. It then parses the logs and convert them into a finite state machine. As more and more logs are utilized and the FSM is built and several 'paths' appear – many that results in completion as expected and those that are taken less often that can lead to errors. The changes that were made by the application along each path are available and these are used to help solve issues later on. The third is the *analytics and monitoring component* that taps into the changes that were made during each stage of the FSM. It is able to calculate the probability of the application reaching completion given its current stage. This helps to monitor the health of the application in real-time and proactively address concerns. It helps understand where exactly the application deviated from expected behavior and what those differences are. The following section delves into each of these components in detail.




### A. Logging Framework

Application logs usually have a few rough guidelines that control the type of content that should be logged. These guidelines vary from company to company and can be quite vague. In most cases, a developer would only log content that he/she feels could help him/her debug the code later on. In this section we put forward a generic XML-formatted design for the logs and strict rules on the content to be logged. The design of the logs are as follows (Fig.1)

```
<L T="Info" D="11/02/2017 12:25:39.460" I="0" AID="00000000-0000-0000-0000-000000000000">
    <FC>
        <F LV="0">Project.Namespace.Class</F>
        <F LV="1">Method</F>
        <F LV="2">Logical Block of Code</F>
    </FC>
    <DC>
        <D Key="Variable Name">Current Value of Variable</D>
        <D Key="Variable Name">Current Value of Variable</D>
    </DC>
    <MC>
        Logical Section of code that is completed
    </MC>
</L>
```

Figure 1. Design of Logs in application

A single log is to be enclosed in <L></L> tags. This tag has multiple attributes which constitute the traditional elements of any logs – the log level (Debug, Info, Warn, and Error etc.) represented by 'T', the date time represented by 'D', the activity ID represented by 'AID' etc. More attributes that remain a necessity for all logs can also be included here. All the content for a log would be within the <L></L> tags. The first section in the logs is called the *'Flow Context'* denoted as 'FC'. Within this section we log the location in the code where this log would be generated from. The section has sub-divisions <F LV="0">, <F LV="1"> etc. Each level helps identify the location from where the log was generated to a greater extent. LV="0" indicates the namespace/project/class while LV="1" indicates the method and so on. More levels

can be added to focus in on the specific location in the code. The next section is '*Data Context'* and denoted by the <DC></DC> tags. It contains all the variables whose values were changed or new variables that were added during the transition from the current state to the next state. Each variable in the Data section is enclosed in Data tags i.e. <D> </D>. These tags have an attribute indicating the name of the variable e.g. <D Key="CustomerNumber">…</D>. The final section is '*Message Context'* and denoted by the <MC></MC> tags. It contains a specific message that helps describe the transition that will take place from the current state to the next stage. Each log would thus help identify the state the application is currently in (using the *Flow Context*) and the transition that caused it to reach that state (using the *message context* and *data context*). This standard of logging helps to create the finite state machine and understand the changes that were made during application processing with greater ease.

### *B. Searching and Creating Finite State Automata*

When issues occur in application processing we begin at the time at which it was noticed and work backwards. For this we use a search framework that can filter through the logs using date-time, log level (error/warn/fatal) etc. Consider the logs shown in Fig.2 as an example. Let say an issue came up at a specific time and we want to see all logs around that time to better understand why the issue occurred. We search and filter the logs to the specific date and time range. The next step is to parse through the logs and create the FSM. Each section of the logs provide certain information.

```
<L T="Info" D="11/02/2017 12:29:39.460" I="0" AID="00000000-0000-0000-0000-000000000000">
    <FC>
        <F LV="0">Project1.Namespace1.Class1</F>
        <F LV="1">Method1</F>
        <F LV="2">Block1</F>
    </FC>
    <MC>
        Validation API called with values - 120, 21/02/02
    </MC>
</L>

<L T="Info" D="11/02/2017 12:29:39.460" I="0" AID="00000000-0000-0000-0000-000000000000">
    <FC>
        <F LV="0">Project1.Namespace1.Class1</F>
        <F LV="1">Method1</F>
        <F LV="2">Block2</F>
    </FC>
    <MC>
        Validation API returned an error
    </MC>
</L>
```

Figure 2. Sample application logs from Order Processing Engine

The Flow context is used to define a state e.g. using the content across levels the states that can be defined are "NameSpace1.Component1-Method1-Block1" and "NameSpace1.Component1-Method1-Block2". The message and data values are used to define a transition e.g. the transitions here are because of "Validation API called with values – 120, 21/02/02" and "Validation API returned an error". The date time of the log (present in the 'D' attribute of parent <L> tags) helps to define the sequence of the states. The rest of the attributes in the <L> tags can be used to make states more information rich.

Creating an FSM provides several advantages. It gives the entire history of application processing – all the actions that took place that would have led to the current state of the application. It also shows all the possible paths the application can follow to reach a desired end state (depicted in Fig.4.).

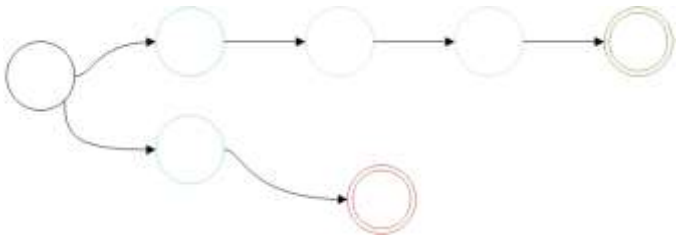


Figure 3. Representation of FSA. Green stage indicates successful application end stats and Red stage indicates an error in application processing.

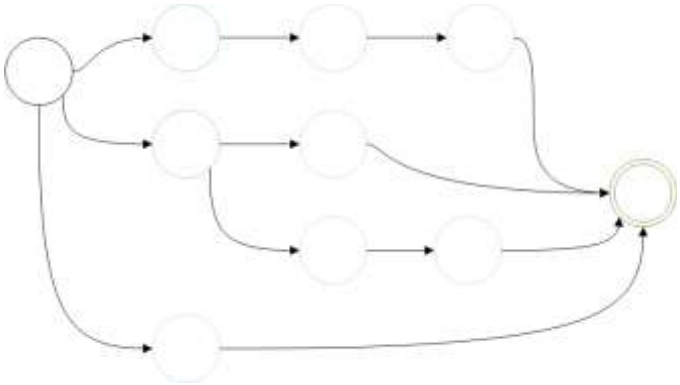


Figure 4. FSA representing application where there are multiple sequence of stages that can lead to successful end state.

Representing the processing of an application as an FSM also helps to identify exactly when the processing of application deviated onto a path that lead to an error. This helps to identify early on when an application might face an error and proactively action on it as shown in Fig.5.

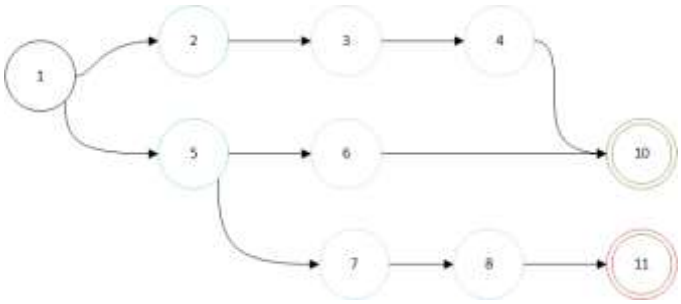



Figure 5. FSA representing application where processing deviates to result in an error-end state. The FSM clearly indicates as soon as application reaches stage-7 it will always go to an error state hence helping to alert stakeholder proactively.

### *C. Monitoring and Driving Decision Making*

Each stage in the FSM will have three attributes - **Total Count, Successful Count** and **Success Rate**. Total Count is the number of the times that stage has been 'reached' i.e. the

number of times a transition has led to the application moving to that stage. For example, the FSM in Fig.5 is the result of four runs of an application: 1-2-3-4-10, 1-5-6-10, 1-5-7-8-11, and 1-2-3-4-10. The resultant count at each stage is as shown in Tab.1. Success count is the number of transitions starting from the current stage eventually ends in the success state. For example in Fig.5 for Stage-1, 3 out of the 4 transition that starting there has eventually led to the success Stage-7. The successful count for the other stages are shown in Tab.1.

The success rate is calculated as given below.

$$\text{Success Rate} = (\text{Successful Count}) / (\text{Total Count}) \quad (1)$$

The success rates for each stage in the FSM represented in Fig.5 has been tabulated in Tab.1.

TABLE I. THE COUNT AND PROBABILITY OF STAGES OF FSA IN FIG.5.

| Stage | Total Count | Successful Count | Success Rate |
|---|---|---|---|
| 1 | 4 | 3 | 75 |
| 2 | 2 | 2 | 100 |
| 3 | 2 | 2 | 100 |
| 4 | 2 | 2 | 100 |
| 5 | 2 | 1 | 50 |
| 6 | 1 | 1 | 100 |
| 7 | 1 | 0 | 0 |
| 8 | 1 | 0 | 0 |
| 10 | 3 | 3 | 100 |
| 11 | 1 | 0 | 0 |

With the success rate being updated in real-time it is possible to monitor the health of the application. When a new log entry causes a transition to a stage where the success rate is less than a fixed threshold, the monitoring module would immediately send out an alert to the stakeholders. This module also help narrow down the exact point at which the application moved to a path where it would most likely end up in an error e.g. In Fig.6 as soon as a log causes the transition of the FSM from stage-5 to stage-7 we can send an alert, causing stakeholder to proactively act on order issues. Finally, since the FSM contains the reasons for transition between stages it can represent in a crisp manner the differences between when the application completed processing as expected and when it does not, as represented in Fig.6.

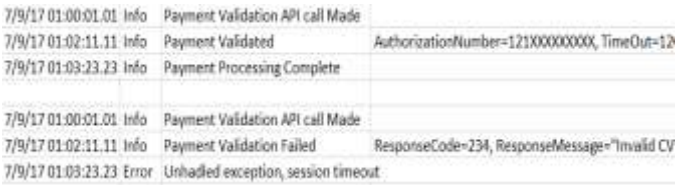



Figure 6. Transition data between stages extracted and represented in easy-to-read format: Date-Time, Message, and Data. The top one depicts normal working and the bottom one represents transition the caused system to error.

The impact of an issue can also be assessed using the Total Count metric e.g. in the FSA depicted in Fig.5 any transition to stage-7 would eventually result in an error. After four runs of the application the Total Count metric is tabulated in Tab.1. Therefore the impact, or percentage of failure is the percentage of times the application transitioned to Stage 7. Using Tab.1 it can be seen that this is equal to (Total Count of Stage-7)/ (Total Count of Stage-1) = 25 percentage. Hence impact can be stated as – "the application is likely to fail 1 in 4 time due to the current issue".

## IV. FURTHER ENHANCEMENTS

The solution discussed has room to evolve by using natural language processing to process the text messages that are recorded during transition. This would help understand topics and sentiments in the logs which can serve as early indicators for issues helping to get more time to act and resolve them.

The decision support system has room to evolve into a prescriptive system, not just providing data and insights but rather being able to prescribe the best course of action to take given the situation in which the application throws an error.

The final state of the system would is to help in self-healing of the application i.e. in situation where the application processing seems to going down a possibly erroneous path it would be able devise a path that would guide the system back to its expected behavior. This involves understanding how such issues were fixed previously and providing the system access to services/API-calls that can make the same changes a human would need to make to fix the issues.

## V. CONCLUSION

The proposed decision support system helps provide data and insights to make decisions on all the main questions for application support i.e. *Why has the issue occurred* – The FSA indicates where the application processing diverged and the transition that caused it to diverge. *How do we solve the issue* – The components help identify the difference between normal expected behavior of the application and the erroneous behavior as depicted in Fig.6. *What is the impact of the issue* – the total count metric helps understand this as explained earlier. In addition the model help to monitor and predict issues prior to when then would occur.

## REFERNCES

[1] Elastic.co, *"Machine Learning-It Catches What You Might Miss, All by Itsel'"*, 2017. [Online] Available : https://www.elastic.co/products/x-pack/machine-learning?utm_source=PredictiveAnalyticsToday&utm_medium=Review&utm_campaign=PAT [Accessed 21-Sept-2017]

[2] Anodot, "Anomaly Detection OEM", 2017. [Online] Available : https://www.anodot.com/product/anomaly-detection-oem/ [Accessed 21-Sept-2017]

[3] Bloomberg, Jason. Intellyx LLC, "Real-Time Anomaly Detection and Analytics for Today's Digital Business", 2016 Intellyx LLC

[4] Prelert, Prelert Inc, "Why Choose Anomaly Detective", 2014. Prelert Inc.

[5] ELKI, "ELKI: Environment for Developing KDD-Applications Supported by Index-Structures", 2017. [Online] Available : https://elki-project.github.io/ [Accessed 21-Sept-2017]

[6] Loom Systems, "Why Modern Businesses Need AIOps?", 2017. [Online] Available : https://www.loomsystems.com/aiops-solution [Accessed 21-Sept-2017]

[7] Numenta, "Machine Intelligence Starts Here", 2017. [Online] Available : https://numenta.org/ [Accessed 21-Sept-2017]

[8] RapidMiner, "RapidMiner Studio", 2017. [Online] Available : https://www.predictiveanalyticstoday.com/rapidminer/ [Accessed 21-Sept-2017]

[9] SciKit-Learn, "scikit-learn : Machine Learning in Python", 2017. [Online] Available : http://scikit-learn.org/stable/ [Accessed 21-Sept-2017]

[10] Weka, "WEKA Data Mining", 2017. [Online] Available : https://www.predictiveanalyticstoday.com/weka-data-mining/ [Accessed 21-Sept-2017]

[11] Interana, "Unleash your inner analytics - SUPERHERO", 2017. [Online] Available : https://www.interana.com/product [Accessed 21-Sept-2017

[12] Rahul Rawlani, 'Application Support & its Business Importance', 2016. [Online] Available: https://www.linkedin.com/pulse/application-support-its-business-importance-rahul-rawlani/ [Accessed 20-Sep-2017]

[13] Splunk Inc, 'Find new answers with Tealeaf and other customer analytics solutions' 2017. [Online] Available: https://www.splunk.com/ [Accessed 27-Sep-2017]

[14] IBM TeaLeaf, 'Accelerate Time to Value' 2017. [Online] Available: https://www-01.ibm.com/software/info/tealeaf/ [Accessed 27-Sep-2017]

[15] AppDynamics LLC US 'Application Performance Management' 2017. [Online] Available: https://www.appdynamics.com/product/application-performance-management/ [Accessed 27-Sep-2017]

[16] WIRED, 'A Short History of DevOps' 2017. [Online] Available: https://www.ca.com/us/rewrite/articles/devops/a-short-history-of-devops.html [Accessed 27-Sep-2017]