NON-INTRUSIVE TRANSACTION MINING FRAMEWORK TO CAPTURE CHARACTERISTICS DURING ENTERPRISE MODERNIZATION ON CLOUD

Ravikumar Ramadoss[1] and Dr.N.M.Elango[2]

[1]Technology Architect, Infosys, Bangalore, Karnataka, India

ravikumar_rrk@hotmail.com

[2]Professor & Head, Department Of computer Applications, RMK Engineering College, Chennai, Tamilnadu, India

nmeoxford@yahoo.com

ABSTRACT

The growth in the popularity of Cloud Computing, proposes to transform the way IT is consumed and managed. With promises from on-demand scaling, Cost effective solutions, faster time to market and innovation at its best introduces the notion of Enterprise modernization of existing applications. While the market is abundant with hype and confusion, the underlying potential is real — and is beginning to be realized and there are lots of on-premise apps running where customers showed their interest of Modernizing. The overall objective of this paper is to provide a methodology to create a non-intrusive transaction mining framework to capture the functional characteristics while modernize their apps to cloud space. This paper proposes a new framework and its architecture to capture the required details from the legacy sources, by adopting the open source utilities, frameworks.

KEYWORDS

ROI, Hybrid clouds, Enterprise Modernization

1. INTRODUCTION

Enterprise Modernization throws a very unique challenge of finalizing the Transactional characteristics for the Critical transactions on the proposed modernization system. The challenge is mainly because of web logs not available. In the existing paradigm, the developers and solution architects propose the transactional characteristics based on their experience from Similar previous assignments and heavily dependent on Business SME to provide the required details. This Paper proposes a solution for the above unique challenge of not having any manual intervention by introducing a non-intrusive log miner to derive Transaction slicing for Enterprise Modernization. Logs (sources data) from different applications gets processed via some defined set of rules and store them in a round robin (or time series) database. The intention of this research paper is to come up with an Open Source Log Mining framework by utilizing all the Open source specifications, tools, and utilities.

2. HIGH LEVEL APPROACH TOWARDS LOG MINING

Log mining uses three high level phases to derive the transaction characteristics. The following diagram explains about the three approaches and the key results achieved out of the three approach

Capture Phase: In capture phase a non-intrusive Aspect oriented component and log distiller is customized to retrieve the pattern of occurrence in the source file. The patterns defined in the Aspects and Distiller is customizable and it can be changed on the fly even when the core transactions are accessed in the legacy applications. A channelized central collection will happen whenever the pattern is recognized on the source files. It emits the data on the available transport channels

Custom Processor Phase: The custom processor is a phase where rules on the retrieved information can be applied for the purpose of filtering or aggregating. The pluggable adaptor part has to be a capability within the processor framework so the results can be derived and persisted to any desired data store. Buffering of the data will happen and then the emitted data are processed based on the defined rule set. Various adaptors are used to monitor and alert the intermediate key characteristics, so that the pattern and the rule sets can be configured dynamically

Storage Phase: The filtered data arrived out the rule set is getting stored on the customized storage model which is then processed and retrieved for Reporting the key Transaction characteristics. Presently, rrd is being taken up for storage purpose and hence an adaptor for the same is in place

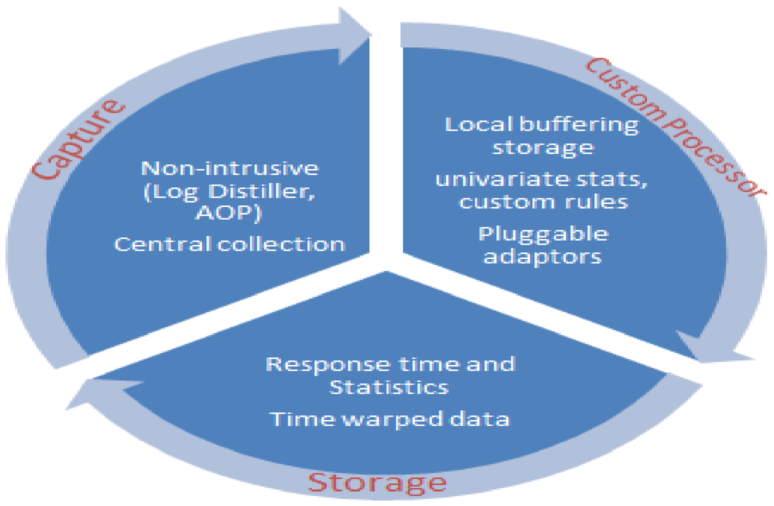


Figure 1. High Level Log Mining Approach

2.1. Log Mining Architecture

The following Architecture is used to retrieve the details required from Legacy Sources. This framework is built with three modular areas: An API layer, a rules processor layer and Storage layer. This corresponds to Logs capture, Logs Processing and Logs Dissemination phases in the mining processes. The API sends these logs characteristics data to a UDP channel with the other end terminated at a Rule processing server. The Rules server would apply defined set of rule to filter, transform or aggregate the transaction characteristics into an RRD store. In addition to this, the API also can possibly send alerts in a nagios fashion (NSCA) based on the metric thresholds that can be set.

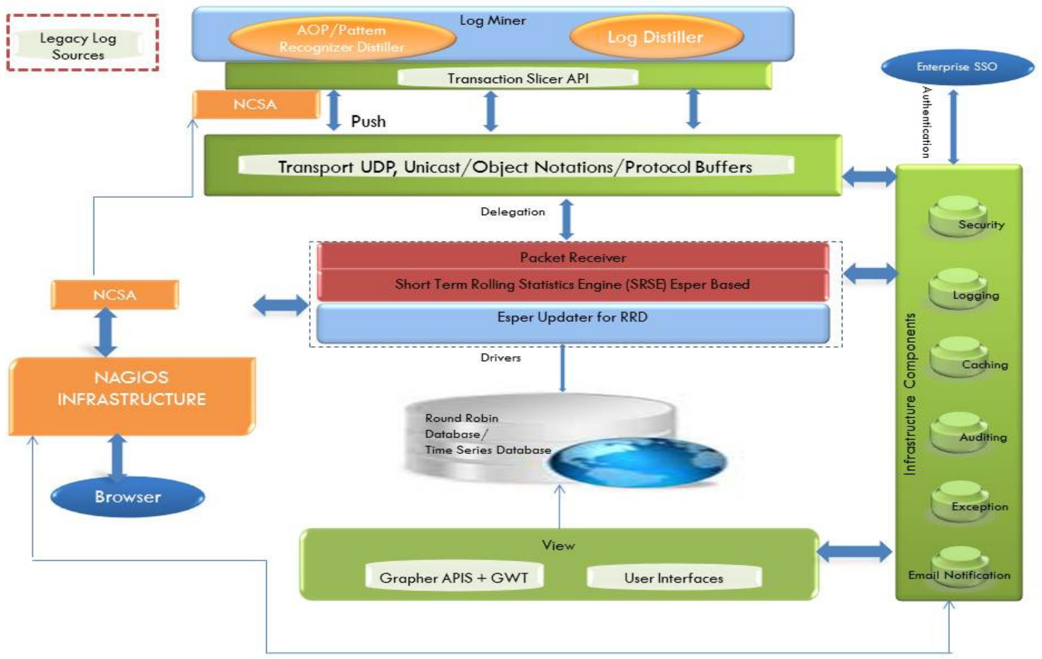


Figure 2. Framework Architecture

The API is deliberately thin just so they can be integrated with other frameworks that specialize on logs mining in different application domains. The configuration changes needed in all the above three modules are kept to minimal for standard use; but yet allow customization to be done when needed.

An “Event” containing log data gets populated with characteristics data and subsequently gets channelled or emitted out to a channel. The different formats of the event got analysed during the POC (Proof of concept) are JSON, Protocol Buffers and LWES along with standard object notation format. The data by themselves could be configured for emitting to different channel based on the need.

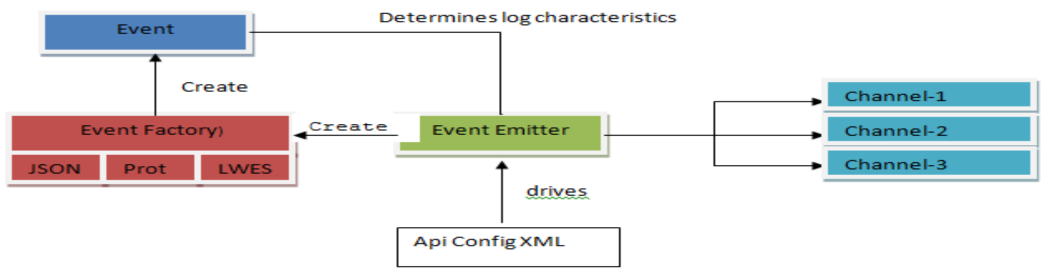


Figure 3. Event channelization

2.2. Event Emitter

The event emitter, once instantiated by the user application can then be used to log (event, value) and then at a subsequent time to emit (). A fresh event is created by the event factory every time an emit is called and the emitters local buffer would be updated into this. The emit function ensures the events are appropriately created for the respective channels and then flushed. A simple snippet of code below explains the façade's simplicity of usage.

```
EventEmitter emitter = new EventEmitter(new EventFactory());
      for (int i = 0; i < 2; i++) {
            emitter.logMetric("responseTime", 12.4 + i);
            emitter.logMetric("requestcount", 10.0 + i);
            emitter.emit();
            Thread.sleep(100);
       }
```

2.3. Event

The event itself created using a factory pattern can be extendible to other formats such as Thrift protocol. Essentially the event representations are serialization protocol dependent and hence are best considered in their respective concrete factories. Event content wise as well could be differently constituted due to protocol vagaries and hence are interface compatible only.

2.4. Channels

The channels are an abstraction that represent differing Medias such as file, network, and console and has two important concepts. One is the media that can also have format in which the event needs to be represented either text or binary. The other is a configured set of metrics that needs to be channelled on it.

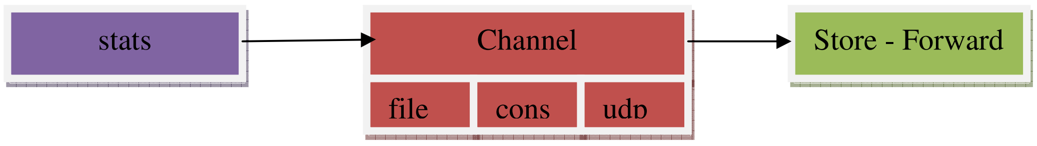


Figure 4. Channel Blocks

The channels do have a flush count to be governed with a strategy to store and forward. The flush count is a policy to control when API can fire actually the packets out; just to control the number emits on the wire. Currently a simple buffer and send strategy is used. The channel block in the below diagram represents abstract channel functions with children classes specializing on only media specifics. If it's a file channel; a file name would be given or if it's an udp channel ip address and port would be given.

2.5. Stats

The typical stats that could be computed for a data could be one of avg, sum, count, max, min, last, first etc. Though the rules server and rrd can possibly do some aggregation; it's always good to have a minimal consolidation capabilities at the API side as the API emits can be considered directly for a rrd at a later stage. As shown in the below diagram; configurability for an application to emit on a channel is extended up to the level individual fine grain stat. As in the example configuration shown below if the metric stat is ResponseTime.Avg it is beamed to channel1; similarly for sum to channel2 and last to channel4. Depending upon the need of flush time for a metric; a channel can be simply defined and attached to a particular stat with a flush period.

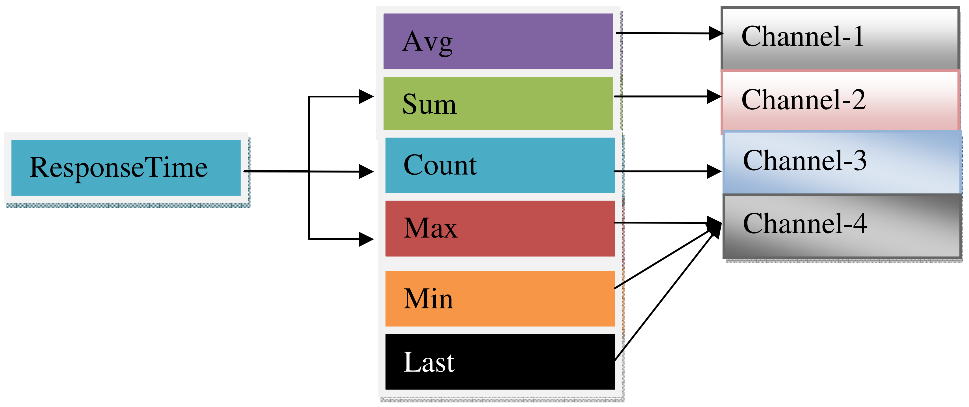


Figure 5. Stats Channelization

2.6. Receivers

The receiver is based on uni/multi cast socket end point that receives and buffers the events into a queue. Specifically an en-queuer thread would be listening on the socket and would receive these into a queue. At the other end of the buffer, is a set of de-queuers that would de-queue the elements of the queue in a handler to do any custom processing. The main entry point of the receiver is located in a class called MulticastEventReciever. This actually provides a listener object to which custom event handlers can be added. In the next sections when we discuss on rules server; it is to be noted that the rules engine naturally extends this multicasteventreceiver to become a server and the handler for the listener would the rules processor specific handling.

2.7 Rules Processor

The metrics once emitted from the API needs to be captured in a rules engine for processing the same to filter, aggregate and emit the results. The nature of metrics data that we derive here are more of time line/series in nature and hence needs a specific treatment in line with its nature as far as rules engines are concerned. The metric emits are more generally considered as event streams with a continuous data coming from a source and a processor that can work on this event streams (on a time or length window) is more appropriate in this context.

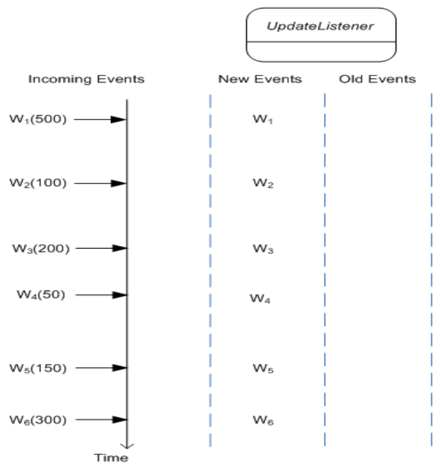


Figure 6. Window Rules Processor

It is important to note on windowing as it is needed to write the esper statements.

If Event has to be filtered on say

Select * from Event (W>=200).win: length (5);

If event with values of W >=200 accepted

Set Length of window Size = 5

Then in the above example W2, W4 and W5 won't even enter the window. So all the events will be still kept in new window if the length is not reached counter (5). Once event containing W>=200 arrives subsequently only then the length (5) is filled up. The old event queue will start filling up once the event doesn't satisfy the window condition.

2.8 Update Listener on RRD

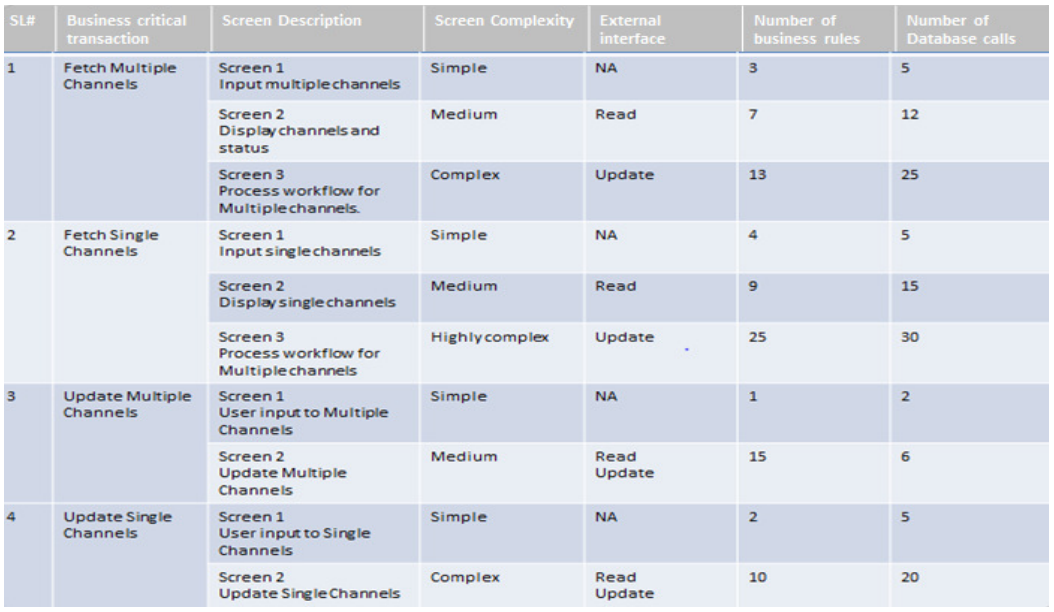
The update listener from esper just needs to implement a call back method which does the following logic:

Select host||'#'||application||'#m1.responseTime' as key, {greatest, least, average, last} as metrics from Event…..

The old Events parameter would be typically null unless the statement has enabled to retain the old events. The new Events are those results that we see in the projection part of the statement as given in the earlier section.

3. Results from Log Miner

Table 1. Channelized Output from Log Miner.



SL#	Business critical transaction	Screen Description	Screen Complexity	External interface	Number of business rules	Number of Database calls
1	Fetch Multiple Channels	Screen 1 Input multiple channels	Simple	NA	3	5
		Screen 2 Display channels and status	Medium	Read	7	12
		Screen 3 Process workflow for Multiple channels.	Complex	Update	13	25
2	Fetch Single Channels	Screen 1 Input single channels	Simple	NA	4	5
		Screen 2 Display single channels	Medium	Read	9	15
		Screen 3 Process workflow for Multiple channels	Highly complex	Update	25	30
3	Update Multiple Channels	Screen 1 User input to Multiple Channels	Simple	NA	1	2
		Screen 2 Update Multiple Channels	Medium	Read Update	15	6
4	Update Single Channels	Screen 1 User input to Single Channels	Simple	NA	2	5
		Screen 2 Update Single Channels	Complex	Read Update	10	20

.

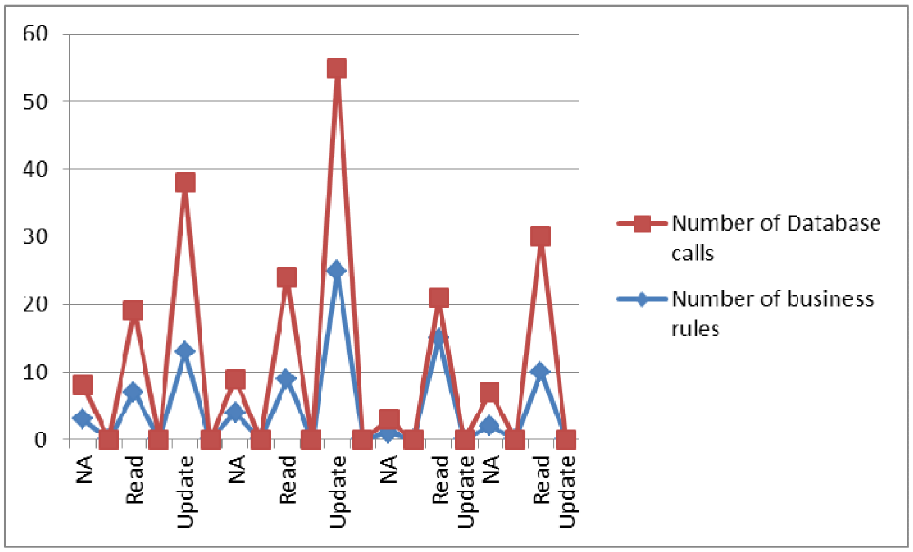


Figure 7. Transaction Slicing

The above table (Table 1) and figure 7 shows the mined details on the characteristics retrieved during enterprise modernization and chart details out the Transaction slicing happened during mining exercise.

4. CONCLUSIONS

This Proposed framework enables application to simply channelize metrics of interests from application to a consolidation server for filtering, aggregating and store the aggregate results on to a rrd. However the framework is composed other open source frameworks (following the policy of re-Use don't re-invent) such as esper and rrd and have plugged in additional code in these areas to make it suitable for achieving the desired result. Therefore it's a hope that this framework be taken to the next level of integration with other frameworks to gain on the values.

To conclude, the proposed framework was developed with an idea to unify metrics channelizing from different legacy applications on simple serialization protocol principles (UDP, Object Notations). However our experiments with AOP have shown that aspects like transaction monitoring, alerting are yet to mature in those frameworks. And also for batch legacy apps there are limited frameworks to capture on service level metrics. Usage of aspects is foundational for automated instrumentation at the runtime.

REFERENCES

[1] The NIST Definition of Cloud Computing; refer to http://csrc.nist.gov/groups/SNS/cloud-computing
[2] Charles H. Fine: Clock Speed – Winning Industry Control in the Age of Temporary Advantage, Basic Books New Edition, 1999, ISBN: 978-0738201535.
[3] Clayton M. Christensen: The Innovator's Dilemma, Harper, 1997, ISBN: 0875845851.
[4] Ivar Subrah: Why Buy the Cow, WebEx Communications, 2007, ISBN: 978-0615163130.
[5] Chris Anderson: Long Tail Future Business Selling, Hyperion, 2006, ISBN: 978-1401302375.
[6] Gizem, Aksahya & Ayese, Ozcan (2009) Communications *& Networks*, Network Books, ABC Publishers.
[7] The NIST Definition of Cloud Computing; refer to http://csrc.nist.gov/groups/SNS/cloud-computing
[8] The RRDTool Definition and Examples; refer to http://oss.oetiker.ch/rrdtool/index.en.html
[9] A. Kivity, "Kvm: The Linux Virtual Machine Monitor," Proc. Ottawa Linux Symp. (OLS '07), pp. 225-230, July 2007.
[10] The Esper complex event processing; refer to http://esper.codehaus.org/.
[11] Nagios, IT Infrastructure monitoring; refer to http://library.nagios.com/
[12] Unicast channelization; refer to http://en.wikipedia.org/wiki/Unicast
[13] Robert Beverly, Karen Sollins, an Internet Protocol Address Clustering Algorithm
[14] Wikipedia - Cloud Computing; refer to http://en.wikipedia.org/wiki/Cloud_computing
[15] Wei-Tek Tsai*, Xin Sun, JanakaBalasooriya; "Service-Oriented Cloud Computing Architecture"; 2010 Seventh International Conference on Information