Execution Trace Streaming based Real Time Collection of Dynamic Metrics using PaaS

Amit Kumar Dogra, Harkomal Singh, Paramvir Singh
Dept. of Computer Science and Engineering
Dr. B R Ambedkar National Institute of Technology, Jalandhar
Punjab, India
amitkumar62003@gmail.com, harkomal.chana@gmail.com, singhpv@nitj.ac.in

***Abstract*—The large amount of data that needs to be processed, still remains one of the major hindrance toward widespread acceptance of dynamic metrics. This paper presents a live execution trace streaming based approach using platform as a service (PaaS) to overcome this hurdle. The proposed approach is the first of its type having the ability to provide the users with live dynamic coupling metric trends. Experiments were carried out on five sample applications with varying size ranging from 100 to 1250 classes. The performance of proposed approach remains to a large extent, independent of increase in application size, number of classes and methods used, number of messages exchanged, and execution time.**

Keywords- Dynamic metrics; Execution trace; live streaming; profiling; dynamic analysis.

I. INTRODUCTION

The class of software metrics that captures the dynamic behavior of software systems are known as dynamic metrics. Static metrics, although very informative and useful for describing overall and structural characteristics of software, however they fail to provide the required accuracy while dealing with intricate features of the software under consideration. This happens due to the presence of runtime features of object oriented software such as dynamic binding, polymorphism, inheritance and unused code. On the other hand, dynamic metrics are computed from the execution trace data of the software. This allows dynamic metrics to capture the impacts of aforementioned runtime features, thereby increasing the accuracy of collected software metrics [1].

The collection of dynamic software metrics involves processing of huge amount of execution trace data. The navigation and exploration of such huge volume of data turns out to be a challenging task and restricts the wide spread adoption of dynamic metrics [1]. This is where the cloud can be very useful as it offers vast reservoir of resources. On the basis of these considerations, we have proposed a distributed dynamic metric evaluation approach based on live streaming of execution trace data using Platform as a service (PaaS) cloud computing model for real time collection of dynamic metrics.

II. PREVIOUS WORK

Chidamber and Kemerer [2] in their research addressed the need for improved software metrics for object oriented softwares. They have defined six static software metrics namely Coupling Between Objects (CBO), Lack of Cohesion of Methods (LCOM), Depth of Inheritance Tree (DIT), Number of Children (NOC), Response for Class (RFC), Weighted Methods for Class (WMC). These six-metrics acted as base for most of the dynamic software metrics that were later introduced by various researchers. Since then, a lot of research has been conducted for evaluation of dynamic metrics [3-10], however to the best of our knowledge none of the previous works focused on the live calculation of dynamic metrics over cloud.

Yacoub et al. [3] defined an improved object level dynamic coupling measures namely, Export Object Coupling (EOC) and Import Object Coupling (IOC) based on executable object-oriented design models. This idea of import and export coupling was further enhanced by Arisholm [4] in which the direction and class both were taken into account. Mitchell and Power [5] further enhanced the work by defining a new set of dynamic coupling metrics on the basis of degree of coupling. Mitchell and Power [6] further studied and verified the relationship between static and dynamic coupling metrics with respect to the influence of instruction coverage.

Another direction to measure dynamic coupling was provided by Hassoun et al. [7] that focused on the measurement of influence that one object imposed on others with respect to time. Zaidman et al. [8] suggested a modification of import coupling and indicated that only some of the coupling metric characteristics were useful for program evaluation. Sarvari et al. [9] have worked on the critical aspects of scale up and speed up while evaluating dynamic metrics using Hadoop MapReduce [11] on cloud.

It was observed during literature survey that, very minimal amount of research has been dedicated toward the benefits of dynamic metrics, and the far-reaching implications it may have on optimization and maintenance of softwares. With this motivation, we propose an approach which can provide us with a means to that end.

III. STUDY DESIGN

In this section, we have presented our research objectives and experimental study design that includes dynamic metrics under consideration, sample applications used for conducting experiments, tool used for profiling the sample applications, methodology employed and system configurations.

A. Research Objectives

We aim to investigate whether PaaS cloud computing model can be used for real-time collection of dynamic metrics via live streaming of execution trace data. For this purpose, we have defined two research objectives:

1) To design and implement a PaaS based dynamic metric collection framework that uses live call streaming.
2) To analyze the performance of the proposed approach using scale-up, latency and response time.

Objective 1 aims to find a way to divide and allocate the trace information to cluster nodes (or gears) so that the metric capturing could be performed in parallel on cloud environment. Parallel computation needs an additional overhead for the distribution of data, data aggregation and data tracking, etc. Objective 2 targets to check, whether the proposed approach makes any positive effect on the performance of metric evaluation with respect to latency, response time and scale-up. Latency is the time taken by the proposed approach between; when a method call is made and when dynamic metric value updating pertaining to that call is complete. However, it is worthwhile to mention that we have ignored latency introduced due to network and only considered latency introduced by the proposed approach. Response time is the time taken between first method call and when collected dynamic metrics are displayed for the first time in DMA tool. Scale-up is the capability of a system to handle increasing amount of trace data with respect to the increasing number of features used and execution time of sample application.

B. Metrics Under Consideration

Using proposed approach, we collected following six dynamic coupling metrics. These metrics were defined by Arisholm in [4].

- EC_CC: Number of distinct client classes using various objects of a given class.
- EC_CM: Number of distinct methods received by all methods of all objects of a given class.
- EC_CD: Number of messages received by all methods of all objects of a class
- IC_CC: Number of distinct server classes used by all methods of all objects of the given class.
- IC_CM: Number of distinct methods invoked by all methods in all the objects of the given class
- IC_CD: Number of messages sent by all methods in all objects of a class.

C. Application Selection Criteria

In previous works, most of the authors have recommended the need of exploring common real world applications for collecting dynamic data. Taking this into cognizance, a holistic set of criteria as presented in Table I, was used for the selection of applications.

TABLE I. APPLICATION SELECTION CRITERIA

S. No.	Criteria	Selection	Remarks
1	Development platform	Java	Most widely used Object Oriented development platform.
2	License policy	Open source	Manipulation of source code and collecting runtime data is easy.
3	User Interaction	GUI based	Most of the contemporary applications are GUI based.
4	Application Size	Varied	Small <10-150>, Medium <151-500>, Large <501-onwards>
5	Nature of Application	Varied	Games, Media players, Graphics tool.

Following the application selection criteria mentioned in Table I, various applications namely JhotDraw [12], SudokuKi [13], XtremeMP [14], GoGrinder [15], Freecol [16] were selected for experimentation.

D. Methodology

The proposed approach is a distributed approach having various modules working in tandem on different machines. In the provided distributed approach, some modules require configurations or setup of environment before actually running to capture dynamic metrics. fig. 1 shows the architecture of the proposed methodology.

E. System Configurations

The local system running the DMA client tool is a machine with 2 GB RAM, intel core 2 duo processor, 320 GB secondary memory and running Ubuntu 14.04.1 operating system. OpenShift [18, 19] PaaS provided by RedHat has been used as cloud environment for implementing the proposed approach. OpenShift PaaS provides flexibility to choose gears that may be of same or different type of architecture. We opted for free account which allows us to use three basic gears, each having single core processor with 512 MB RAM, and 1GB storage. These three gears can work like a cluster and efficiently process large amount of data.

IV. TOOL SUPPORT

In order to implement the proposed approach, a supporting tool namely Dynamic Metric Analysis (DMA) [20] has been developed. It has been developed purely in Java language using JP2 [17] as an instrumentation module. However, the proposed approach customizes JP2 to segregate each method call instead of making a calling context tree (CCT), so that it can be streamed live to the cloud. OpenShift [18, 19] cloud is used to provide the needed cloud platform. In this section, various modules of DMA tool, namely configuration module, customized JP2 module, Server module, and Cluster module have been explained.

A. Configuration Module

This module allows the user to specify the executable java archive (.jar) files of the application to be analyzed along with any other supporting jar that may be required for its execution.

These supporting jars are however not instrumented. The user is also required to specify the localhost address and port number with which the cloud gears have been linked. Since the trace data generated during the execution can be huge, an option of manually balancing the load across the gears has also been provided.

B. Customized JP2 Module

This module is responsible for instrumenting the byte code of the application to be analyzed. Inherently JP2 is a calling context profiler for java virtual machine which has the ability to collects complete and accurate profiles. However, for implementing the proposed approach, we have done away with the dumping part of JP2, which generates the CCT. Instead, JP2 has been customized to send trace data of each method call as and when it happens at runtime to server module.

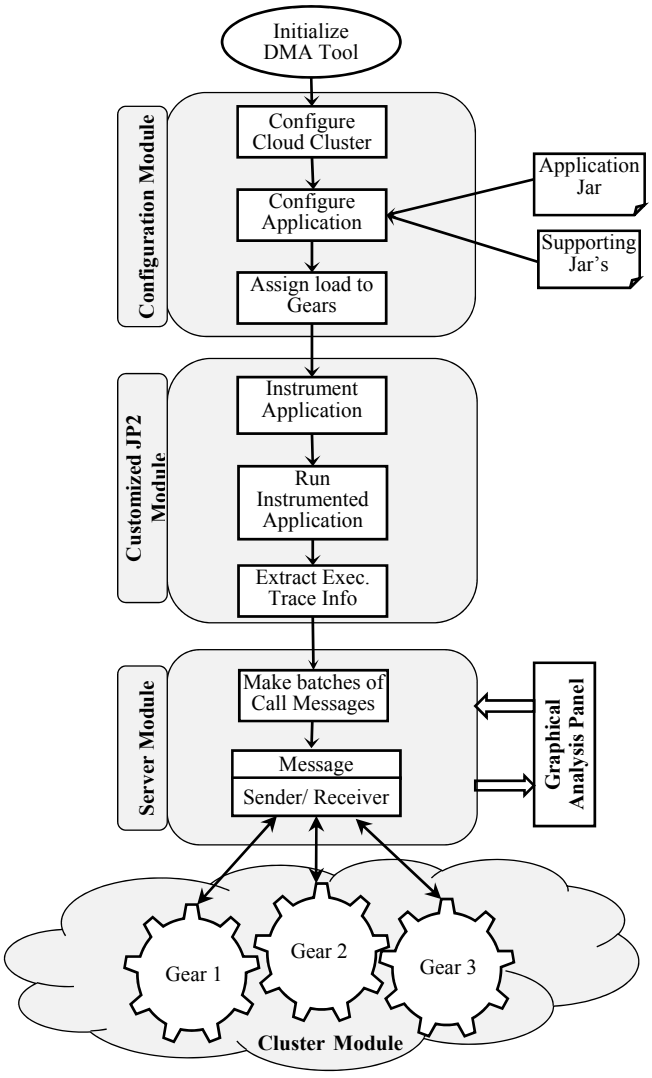

Figure 1. Architecture of proposed approach

C. Server Module

Server module acts as a bridge between the cluster module (cloud gears) and DMA. Server module receives the trace data as messages from instrumented classes and sends them to corresponding cluster gears on cloud. Server module provides the following facilities:

- Server sends a READY signal to each gear and waits for ACKNOWLEDGMENT signal from gears as response. If any of the gear is unable to respond back in predefined period that gear will be assumed not ready for communication.
- Retrieve the cluster load information from configuration module and distribute corresponding messages to gears.
- Sends the batch of messages to corresponding gears.
- Receives the updated dynamic metrics from cluster module and store locally.
- Send CLEAR cluster request to each of the gear while starting new metrics capturing process.

D. Cluster Module

This module resides on cloud gears. This Module communicates only with server module and exchanges following information with it:

- Receives READY signal from sever module and respond back with an ACKNOWLEDGMENT to signal the server module of DMA client that this gear is up and ready for communication.
- Receives batch of method call messages from server module and processes them to compute various dynamic metrics and finally send it back to server.

```
Process_Batch <batch> {
    For Each <msg m> in batch {
        Extract caller class<Cr> and caller method<Mr>
        Extract callee class<Ce> and callee method<Me>
        Update Import coupling metrics for Cr
            IC_CC, IC_CM, IC_CD
        Update Export coupling metrics for Ce
            EC_CC, EC_CM, EC_CD
        Mark Cr and Ce as updated
    }
    Prepare Batch_Reply_Msg
    Append metric values of all classes marked updated
    Send Batch_Reply_Msg to server module
}
```

Figure 2. Pseudocode of cluster module

E. Graphical Analysis Panel

The DMA tool has been provided with a very dynamic and user friendly graphical analysis panel as shown in fig. 3. Graphical analysis panel has three sub panels:

- Top panel shows the list of classes along with their import and export coupling metric values for that particular scenario.
- Bottom-left panel shows the coupling (EC_CD and IC_CD) of the class selected from the top panel with

- Bottom-right panel shows the change in the import and export coupling metric value for the selected class over a fixed time interval. This feature becomes very useful when the user wants to study the change in coupling metrics for using any particular feature of the application being analyzed

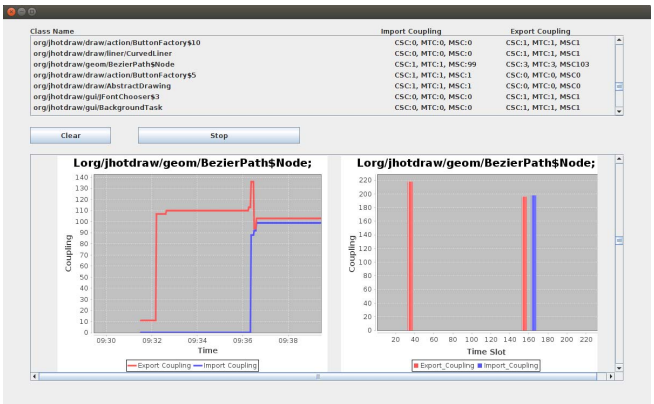

Figure 3. Graphical Analysis panel

The DMA tool also provides the user with an option to export the dynamic coupling metric values (vis-à-vis EC_CC, EC_CM, EC_CD, IC_CC, IC_CM and IC_CD) captured, in excel format which enables the user to save and perform other detailed investigation of dynamic metrics

V. PERFORMANCE ANALYSIS

Since the proposed methodology follows a distributed approach, so latency and response time are major performance attributes. We analyzed the performance of proposed approach and the developed DMA tool for these two parameters with respect to increasing application size, number of classes and methods used during execution, number of messages exchanged during execution and execution time.

A. Application Size

With the application size, complexity and resource requirements also increases. We have used two measures of application size namely Line of Codes (LOC) and number of classes in the application. From fig. 4 and fig. 5, it is evident that our approach is inert to increase in size of application both in terms of lines of codes as well as number of classes. However, we would like to point out that during experimentation only latency introduced by our approach has been considered while the latency incurred due to network has not been considered.

TABLE II. PERFORMANCE ANALYSIS OF DMA TOOL

Application	LOC	# Classes	Experiment No.	# Classes Used	# Methods Used	# Messages Exchanged	Execution Time(s)	Latency(ms)	Response Time(ms)
SudokuKi	6464	126	1	52	133	489	634	374	1978
			2	64	142	782	648	382	1875
			3	68	154	976	716	378	1954
			4	76	158	1164	759	391	1986
			5	84	192	1246	897	385	1992
			Average					382	1957
XtremeMP	10202	152	1	83	184	978	614	438	1989
			2	102	257	1226	752	429	1973
			3	104	269	1374	846	433	1986
			4	116	304	1586	968	428	1993
			5	127	348	1858	983	440	2007
			Average					434	1990
GoGrinder	9120	163	1	98	172	1092	337	462	1964
			2	105	185	1186	412	458	1973
			3	118	216	1374	408	431	1982
			4	139	264	1504	617	457	2025
			5	155	268	1482	852	456	2017
			Average					453	1992
JHotDraw	82270	388	1	135	345	13662	479	467	1997
			2	157	371	19640	658	465	1996
			3	174	368	24620	674	457	2153
			4	191	497	23876	1247	454	2023
			5	218	634	29468	1342	470	2048
			Average					463	2043
FreeCol	163595	1244	1	584	3273	31794	876	491	2034
			2	612	3569	39682	986	489	2017
			3	694	4291	46488	1328	512	2084
			4	658	3978	43926	1410	504	2138
			5	768	4357	48762	1602	517	2089
			Average					503	2072

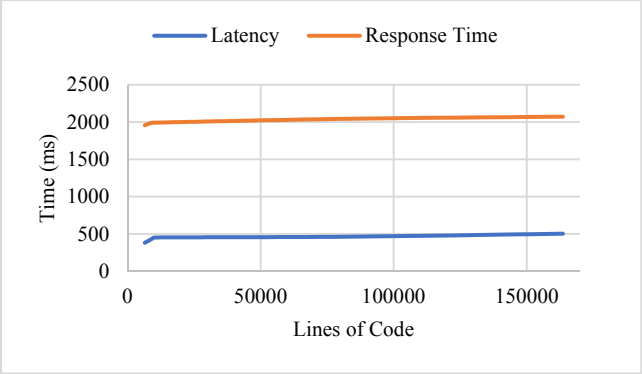

Figure 4. Latency and Response Time Vs LOC

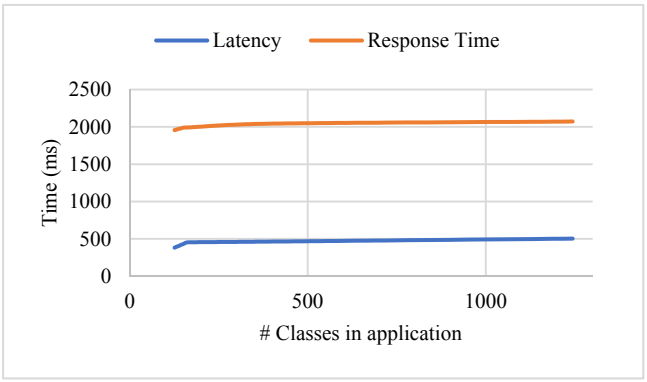

Figure 5. Latency and Response Time Vs # Classes

B. *Number of Classes Used*

Although the de facto measure of application size is LOC and number of classes, however they seldom present the right picture as all classes defined or all the lines of code are rarely used during execution, thereby we analyzed the latency of our approach against the number of classes actually used. So, we conducted experiments with selected applications by continuously increasing the number of features used in every round of execution, thereby varying the number of classes used during application run. The results of the same have been presented in fig. 6.

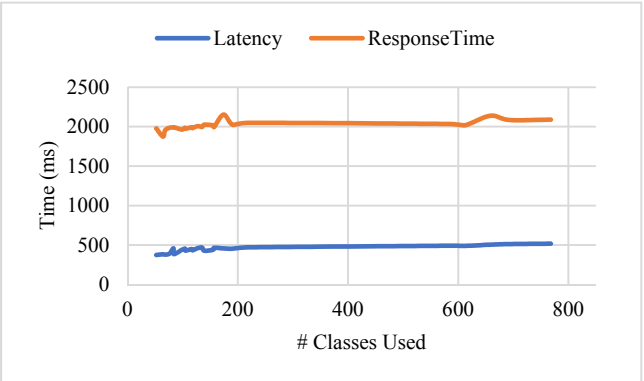

Figure 6. Latency and Response time Vs # Classes used

C. *Number of Methods Used*

Fig. 7 below shows the performance of proposed approach with increasing the number of distinct methods called during the execution of the applications.

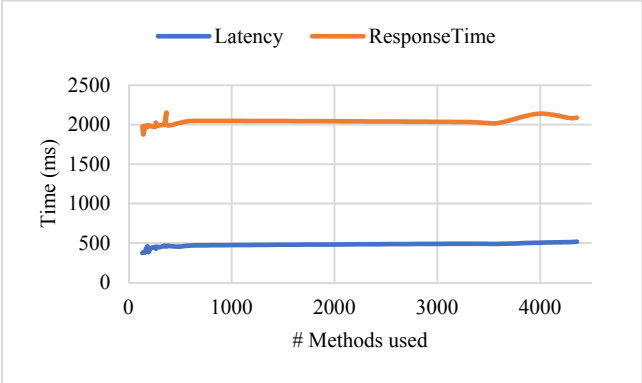

Figure 7. Latency and Response time Vs # Methods used

From fig. 7, it can be easily visualized that increase in number of distinct methods called does not impact the latency and response time of our approach.

D. *Number of Messages Exchanged*

In this section, we analyzed our approach against the increasing number of messages exchanged during the application run, where each message corresponds to a method call.

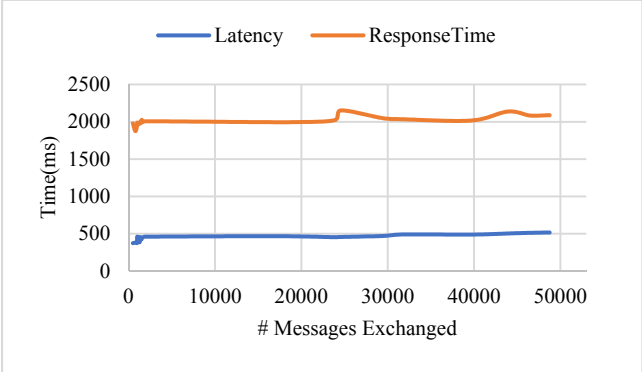

Figure 8. Latency and Response time Vs # Messages exchanged

From fig. 8 it can be easily observed that increase in number of messages has relatively no impact on latency and response time of the approach. This observation is very important as it supports our claim that the proposed approach handles speed up without any change in its performance and efficiency.

E. *Execution Time*

As our final measure for performance analysis of proposed approach, we investigated the relationship between Latency and response time with the execution time of the sample application. Larger execution time generally corresponds to greater number of features used during execution. Moreover,

with the increase in execution time, the fluctuations caused due to network also tends to normalized.

Figure 9. Latency and Response time Vs Execution time

Fig. 9 shows that both latency and response time are independent of the increase in execution time of the application being analyzed.

VI. Threats to Validity

The results and observations presented in this paper cannot be generalized to other applications since they face several threats to validity. One of the major threat is high dependence of dynamic metrics on inputs provided to the system under consideration, so the observations become very hard to be repeated. Lack of well-defined test cases for majority of open source applications proves to be another hurdle in validating the observations. Furthermore, our approach presumes a fast and reliable internet connection, which may not be the case always. Although the applications under consideration have been selected carefully, but they may not be a proper representative of real world applications.

VII. Conclusions and Future Scope

In recent years, researchers have shown great concern over obstruction caused by scalability issues in collection of dynamic metrics. Performance issues caused due to large amount of execution trace data are major problem in metrics validation. The logical solution to this issue is to use cloud (PaaS) as it will provide us with ample resources and processing power to capture dynamic metrics.

The primary objective of this research work has been accomplished by designing and implementing an approach that process huge amount of trace data in parallel on cluster of cloud gears for live dynamic metric evaluation. Based on the results presented in Section V, it is evident that performance of proposed approach remains to a large extent, independent of increase in application size, number of classes, number of methods used, messages exchanged, and execution time. In summary, it is fair to say that the proposed approach (and DMA tool) is the first tool of its type, having the ability to provide the users with live dynamic metrics. This capability of the tool to capture metrics for a particular event in the application can prove to be immensely useful.

The results observed in this work provide ample scope for future research. Firstly, in this work only six dynamic coupling metrics were captured, so the tool lacks support for a complete dynamic metric suite. Thereby the proposed approach can be further enhanced to include all the dynamic coupling and cohesion metrics in future. Secondly, considering the continuous drift of software industry from desktop applications to mobile and web applications, the proposed approach should be further developed to analyze such applications.

References

[1] R. D. Venkatasubramanyam and G. R. Sowmya, "Why is dynamic analysis not used as extensively as static analysis: an industrial study," In Proceedings of the 1st International Workshop on Software Engineering Research and Industrial Practices, 2014, pp. 24-33.

[2] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol. 20, 1994.

[3] S. M. Yacoub, H. H. Ammar and T. Robinson, "Dynamic metrics for object oriented designs", In International Symposium on Software Metrics, Boca Raton, pp. 50 – 61, 1999.

[4] E. Arisholm, L. C. Briand and A. Foyen, "Dynamic Coupling Measurement for Object-Oriented Software", IEEE Transactions on Software Engineering, vol. 30, pp. 491-506, 2004.

[5] A. Mitchell and J. F. Power, "Using object-level run-time metrics to study coupling between objects", ACM Symposium on Applied Computing, pp. 1456-1462, 2005.

[6] A. Mitchell and J. F. Power, "A study of the influence of coverage on the relationship between static and dynamic coupling metrics", Science of Computer Programming, vol. 59, no. 1/2, pp. 4-25, 2006.

[7] Y. Hassoun, S. Counsell, and R. Johnson, "Dynamic coupling metric: proof of concept", IEEE Proceedings-Software, vol. 152, no. 6, pp. 273-279, 2005.

[8] A. Zaidman, S. Demeyer, "Analyzing large event traces with the help of coupling metrics." 5th International Workshop on OO Reengineering, Oslo, Norway, 2004.

[9] S. Sarvari, P. Singh and G. Sikka, "Efficient and Scalable Collection of Dynamic Metrics Using MapReduce," Asia-Pacific Software Engineering Conference (APSEC), pp. 127-134, 2015

[10] S.G. Tahir and A. Macdonell, "A Systematic Mapping Study on Dynamic Metrics and Software Quality", 28th IEEE International Conference on Software Maintenance, Trento, pp. 326 - 335, 2012.

[11] Apache Hadoop Map Reduce. http://hadoop.apache.org/mapreduce/

[12] http://www.jhotdraw.org/

[13] http://sudokuki.sourceforge.net/

[14] https://sourceforge.net/projects/xtreme/

[15] http://gogrinder.sourceforge.net/

[16] http:// http://www.freecol.org/

[17] A. Sarimbekov, A. Sewe, W. Binder, P. Moret and M. Mezini, "JP2: Call-site aware calling context profiling for the Java Virtual Machine," In Science of Computer Programming, vol. 79, pp. 146–157, 2014

[18] OpenShift online application management portal http://www.openshift.com.

[19] https://developers.openshift.com/managing-your-appplications/client-tools.html

[20] https://github.com/pv-singh/DMA