Journal of Theoretical and Applied Information Technology
20th June 2014. Vol. 64 No.2
© 2005 - 2014 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN APPROACH BASED ON BAND TO PREDICT FAULT LOCALIZATION IN CLOUD ENVIRONMENT

NETHAJI[1], CHANDRASEKAR[2]

[1] Research Scholar, Computer Science department, Karpagam University, Eachanari Post, Coimbatore, Tamilnadu, India.

[2] Associate Professor, Computer Science department, Periyar University, Salem, Tamilnadu, India.

E-mail: [1]nethaj.babu@gmail.com, [2]ccsekar@gmail.com

ABSTRACT

Despite the advancement in software tools and processes, bugs are prevalent in many systems. There is a need to develop a system with automated means to help reduce software debugging cost. One important confront in debugging is to restrict the root cause of program failures. GenProg uses an extensive structure of genetic programming to develop a program variant that retains essential functionality but it is not vulnerable to a known deficiency. The existing software testing suite identifies program faults in cloud environment. Structural differencing algorithms and delta debugging minimizes the dissimilarity among variant and the original program in terms of minimum repair. GenProg are not more sophisticated with the localization technique and ranking are not performed with different acceptable patches in software testing. Subsequently, Fault Localization based on Band (FLB) mechanism is introduced to overcome the faults and rank the different acceptable patches. Fault Localization based on Band outputs an ordered list of program elements sorted based on their likelihood. Based on the likelihood, the root cause for a set of failures is identified in cloud environment. Band based fault localization extracts the number of features in standard cloud that are potentially associated to the usefulness of fault localization in software testing. It builds machine learning process and these feature values discover out a discriminative model that is significant to predict the fault localization and effectiveness in ranking. An experimental evaluation is carried out with the Amazon EC2 dataset to estimate the performance of the proposed FLB mechanism with GenProg. Performance metric for evaluation of FLB is measured in terms of CPU utilization, percent time overhead, communication cost, average auditing time, normalized throughput and performance counter.

Keywords: *Fault Localization, Amazon EC2 dataset, Genetic programming, Software Testing, Machine Learning process, Discriminative model, Structural Differencing Algorithms.*

1. INTRODUCTION

Despite the progression in software tools and processes, bugs are widespread in many systems. Thus there arises a need to automate means that help reduce software debugging cost. One important challenge in software debugging is to confine the root cause of program failures in cloud zone.

When a software program fails, it is frequently rigid to place the faulty program elements that are dependable for the failure. Nefeli, a virtual infrastructure gateway in [4] lifts the restriction and cloud consumers afford deployment hints on the achievable mapping of VMs to physical nodes. The hints in [4] include the collocation and anti-collocation of VMs, the survival of potential performance bottlenecks, the existence of underlying hardware features, but the proximity of certain VM fails to address scalability issues present in large infrastructures.

Iterative pattern mining without any sub pattern have the same support. Iterative generators as demonstrated in [2] is paired with closed patterns to construct a set of rules expressing forward, backward, and in-between temporal constraints surrounded by events in one normal representation. The utility of the iterative pattern mining fails to perform the nested pattern data. The root cause could be positioned far from the location where the software failure is exhibit, e.g., the location where a program crashes or produces a wrong output in cloud environment.

In order to address the high cost of software debugging in general, and help in localizing root causes of failures in particular, many software localization tools in cloud have been anticipated. These tools typically take in a set of normal execution traces and another set of faulty execution traces. Based on these set of software program execution traces, these tools allocate dishonesty scores to various program elements. Next, software program elements could be sorted based on their dishonesty scores in descending order. The resultant list of software program elements can then be presented to a human debugger to aid in finding the root cause of the set of failures.

Collaborative provable data possession scheme as shown in [3] uses the techniques of Homomorphic demonstrable responses and hash index hierarchy. Collaborative fails to expand more effective and practical CPDP constructions. First performance of CPDP scheme, especially for large files, is seriously affected by the bilinear mapping operations because of high complexity. Cooperative PDP (CPDP) scheme proves the security based on multi prover zero-knowledge proof system in [10], which assure unity bit but it is affected by the bilinear mapping operations due to its high complexity. Additionally, articulate performance optimization mechanisms for CPDP scheme present an efficient method for identifying the parameter values to reduce the cost involved during computation of clients and storage service providers.

Hierarchical Attribute Set Based Encryption (HASBE) extended cipher text-policy Attribute-Set-Based Encryption (ASBE) with a hierarchical structure of users. The ASBE scheme as shown in [15] not only attains scalability due to its hierarchical arrangement, but also inherits elasticity and fine-grained accesses manage in supporting compound attributes of ASBE. ASBE efficiently share confidential data on cloud servers using Hierarchical Identity Based Encryption (HIBE) system and the Cipher Text-Policy Attribute-Based Encryption (CP-ABE) system, and finally providing performance expressivity trade off as described in [18].

Integrating key feature of Hierarchical Attribute Based Encryption (HABE) and Cipher Text Policy Attribute Based Encryption (CP-ABE) system as shown in [19], did not achieved high performance and fine grained access rate. These error occur when user revocation scheme have no longer use of organization. Secure outsourcing mechanism for solving large-scale systems of Linear Equations (LE) in [13] applies LU decomposition to such large-scale LE would be prohibitively expensive, building the secure LE outsourcing mechanism via a wholly different approach. Iterative method is much easier to execute in practice and only demands comparatively simpler matrix-vector operations.

As the necessary bread and butter of data forensics and post investigation as shown in [7], is characterized by providing with the information privacy. The privacy is provided on sensitive documents stored in cloud, unidentified authentication on user access, and provenance tracking on doubtful documents. Preserving the privacy of intermediate datasets as shown in [17] becomes a demanding problem since adversaries recover privacy sensitive information by inspecting multiple intermediate datasets. Upper-bound privacy leakage restriction based approach recognizes intermediate datasets which are encrypted. As a result, the privacy-preserving cost is saved whereas the privacy necessity of data holders is not satisfied.

Fine-grained access control as shown in [20] assures the privacy of the data from the cloud and preserves the privacy of users who are authorized to access the data. Data access scalability and fine-grained process is achieved using PHRs, leverage encrypts each patient’s PHR file as depicted in [12]. The multiple data owner scenarios and dividing the users in the PHR system into several security domains very much reduce the key management complexity for owners and users.

Cloud scheduler which takes into consideration the two types of requirements, user and infrastructure in hand fails to focus extending trustworthy collection/calculation of the other properties. The trust measurements performed by the DC-C fails in identifying the building up resource’s RCoT and its integrity measurements in [6]. The problem of assigning a Third Party Auditor (TPA) as illustrated in [5, 8] proves the model of integrity for dynamic types of data stored in the cloud. The foreword of TPA eliminates the connection of the client through the auditing of whether data stored in the cloud are definitely intact, which is important in achieving maximum scale for Cloud Computing. The most promising one is a model in which public verifiability is enforced and does not allow TPA to audit the cloud data storage without difficult users’ time, probability or resources.

Certain types of high decentralized information accountability framework keep following the definite handling of the users’ data in the cloud. In particular, object-centred approach [9] that enables surround logging mechanism together with users' data and policies fails to confirm the integrity of the JRE and the authentication of JARs. A decentralized mechanism for such self-adaptation as shown in [14], using market-based heuristics does not enrich CloudSim. Decentralized mechanism also fails in methodically at these junctures to observe their consequence on the combined adaptation.

Greedy heuristics solve the online problem as illustrated in [11] has numerous natural applications in the circumstance of self-determining tasks scheduling or virtual machines allocation. Resource allocation considers both the offline and the online versions of the problem. The degree constraint on the maximal number of clients serves a realistic form in many contexts. Statistical Process Control (SPC) charts as exemplified in [16] identify performance anomalies and differential profiling to classify their root causes. By automating the tasks within the framework fails to expand the scope of automation, based on detailed analysis of profiling data. Profiling data includes report generation of probable culprits and expect to find other areas of software development.

An effective fault localization mechanism would return a root cause using the apprehensive list program elements. Although existing method with fault localization is effective only on some of the cases, regrettably, for many other cases, fault localization method are not effectual sufficient. GenProg as shown in [1] is an automated method for repairing defects in off-the-shelf, legacy programs without official condition, program annotations, or particular coding practices. GenProg uses an extended form of genetic programming but not sophisticated with the localization technique and ranking. Structural differencing algorithms and delta debugging decrease the difference between this variant and the unique program to a least repair. Root causes are often listed low in the record of most distrustful program elements. The unreliability of fault localization tools potentially motive many developers to distrust fault localization methods.

Fault localization takes as effort of a faulty program, along with a set of test cases. The faulty program is instrumented such that when a test case is run over it, a program band (i.e.,) in the form of spectrum is generated. A program band records certain characteristics of a particular program run and thus it becomes a behavioural signature of the run. The program band constitutes a set of counters which record how many times different program elements are executed in a particular program run. Alternatively, the counter could record a Boolean standard that indicates whether a program element is executed or not. Fault localization task is to examine program band of correct and faulty runs with the program elements for effective performance counter.

Based on the aforementioned methods and techniques, an FLB mechanism is presented, plan to increase the usability of fault localization model by building a system to predict if a particular output of a fault localization model is likely to be effective or not. An output of fault localization model is effectual if the faulty program component or root cause is listed among the program elements. With FLB model, the debuggers is better notified whether the output trusts of fault localization run on a set of program execution traces. The FLB mechanism, contributions define a new research problem by ranking the different acceptable patches. Solving the GenProg problem would help developers to better trust the output of a fault localization model based on band. A machine learning framework tackles the research problem by ranking with different acceptable patches. The set of features are appropriate for predicting the usefulness of a fault localization model.

The structure of this paper is as follows. In Section 2, describe preliminary materials on fault localization and the diverse form of existing work with their limitations. In Section 3, present an eye view of FLB mechanism by outlining the features extracted from the execution traces and output of the fault localization model. Section 4 and 5 outline experiment settings, datasets, and present results which answer a number of research questions. Section 6 finally concludes with helpful solution.

2. BAND BASED FAULT LOCALIZATION MECHANISM

The goal of Fault Localization based on Band (FLB) mechanism is to build a model that predict fault in an effective way. To realize FLB, illustrated in Figure 1.1, Amazon EC2 dataset are taken for the fault localization in cloud environment. Information from Amazon EC2 dataset is leveraged to predict fault localization model with different set of program execution traces with the fault being localized based on the band. Band in FLB mechanism depends on the spectra that contains ordered list of program elements. The ordered list of program elements in FLB mechanism are sorted based on the likelihood. Fault Localization based on Band extracts features in cloud that are potentially associated to perform effective ranking in cloud environment.

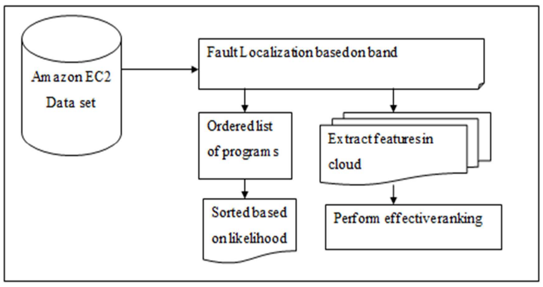
Amazon EC2 Data set
Fault Localization based on band
Ordered list of programs
Extract features in cloud
Sorted based on likelihood
Perform effective ranking

Figure 1.1 Overall Architecture Diagram of FLB Mechanism

In the special case, as depicted in figure 1.1, where all program elements are given the same dishonest score values, there is a very low likelihood that the fault localization model is effective for those execution traces. Fault localization comprises of two phases namely instruction phase and operation phase. The instruction phase output a model that distinguish effective and ineffective fault localization instances. The operation phase applies to a number of unknown fault localization instances and output if the cases are probable to be effective. Figure 1.2 and Figure 1.3 describe these two phases in more detail.

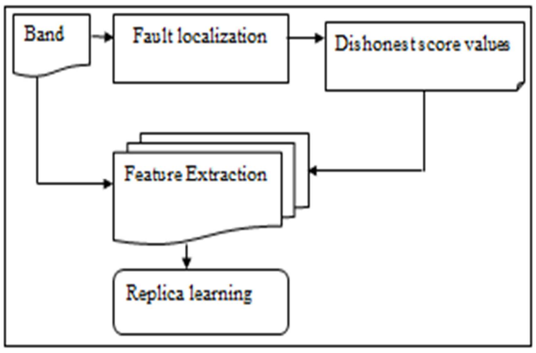

Figure1.2 Instruction Phase

During the instruction phase, a set of fault localization occasions are taken into account. Some of these cases are effective and some others are ineffective. The instruction phase executes two processes namely feature extraction, and replica learning. During feature extraction, based on an instruction data, the feature values that shed light into certain imperative characteristics are extracted that potentially distinguish between the effective and ineffective instances. In the replica learning process, the feature values of each of the instruction instances along with the effectiveness tag are used to build a discriminative model which forecast whether an unknown fault localization illustration is effective or not. Each of these cases is represented by the program band corresponding to correct and faulty execution traces. A list of dishonest score values are assigned by the fault localization model to the program elements. An effectual tag is assigned if the root cause is of top five or it is unsuccessful. This discriminative model is output to the operation phase.

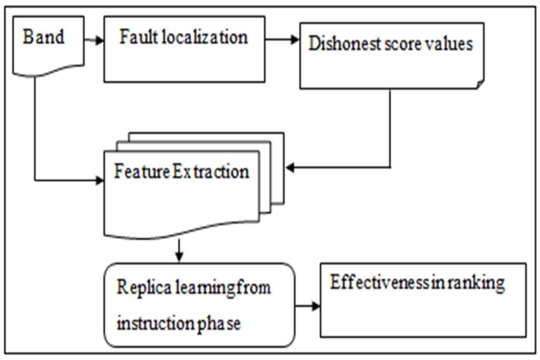

Figure 1.3 Operation Phase

The operation phase consists of feature extraction and effectiveness in ranking. Moreover, feature values are extracted from indefinite instances whose tags, effective or ineffective, are to be ranked. These values are then fed to the discriminative model learned in the instruction phase. The model would then output a ranking.

2.1 Instruction and Operation Phase Feature Extraction

Instruction and operation phase takes the feature extract values from input execution traces and from the outputs fault localization model. Fifteen features are extracted from the FLB input execution traces and the remaining thirty features are extracted from the dishonest score values output. Let us consider a scenario with fifteen input features ‘P1’ to ‘P5’ (traces) and ‘E1’ to ‘E10’ (elements) that capture information about program execution traces and program elements covered by these execution traces. Features from ‘P1’ to ‘P5’ capture information available for fault localization.

Lesser number of trace points in cloud environment might cause effective fault localization performance.

Features 'E1' to 'E4' capture the information on program elements that are covered by the execution traces. Higher the number of program elements, the more simplicity to localize the faults and likely to compare and differentiate the elements. With more program elements, the faulty program element is assigned with the same or lower dishonest score values than the other program elements for easy diagnosis Feature 'E5' in input execution traces captures certain program elements that appear in faults. Feature 'E6' captures the opposite which indicate omission errors where some program elements have to be executed. Features 'E7' to 'E10' capture the two highest proportions of failures provided by one program element. Intuitively, the higher the proportion of failures that passes a program element, the more likely it is the root cause.

The next thirty output features capture the dishonest scores from FLB mechanism. Output features of FLB from 'D1' to 'D10' capture the top dishonest score values. If the dishonest score value are too low, intuitively it is less likely for a fault localization instance to be effective. Features 'S1' to 'S6' evaluate simple information of the top 10 dishonest score values. FLB mechanism with 'S' series serve as information summary of the score. Features 'B1' to 'B11' and 'C1' to 'C3' are aimed to capture a break and perform relative difference in the top 10 dishonest score values. The break from 'B1' to 'B11' is able to localize the faults in FLB and differentiate some program elements to be significantly more dishonest score than the others. This might indicate that some of the top 10 program elements are probably to be the root cause and able to differentiate elements in effective way.

2.2 Instruction Phase Replica Learning

The instruction phase serves as the inputs to replica learning from feature extraction, set of instruction instances with their effectiveness tags. Each of the instances is represented as 40 feature values produced by the feature extraction process as described in Section A. The goal of the replica learning process is to convert these set of feature vectors into a discriminative model that could predict the effectiveness tag of a fault localization instance whose effectiveness is unknown. The chosen Utmost Subsidiary Hyperplane (USH) separates two classes of information (i.e., fault and faultless). For example, consider an instruction phase with Amazon EC2 dataset in form of

$$\vec{p}_i, q_i \qquad \text{Eqn (1)}$$

Where, $\vec{p}_i$ is the feature vector of the i^{th} instruction data instance and q_i represents tag of data instance ($q_i \in \{+1, -1\}$). The problem of searching for a separating Hyperplane with utmost subsidiary is reduced to finding the minimal value. The minimum value is represented as

$$\frac{1}{2}|\vec{Z}| = \frac{1}{2}\sqrt{Z_1^2 + Z_2^2 + Z_n^2} \qquad \text{Eqn (2)}$$

Which, satisfies the constrains,

$$q_i(\vec{Z}.\vec{p}_i + c) \geq 1 \qquad \text{Eqn (3)}$$

Where, $\vec{Z}$ is perpendicular to the separating Hyperplane and n is the number of attributes and c is a constant number indicates position of the Hyperplane in cloud space.

2.3 Operation Phase of FLB for effective ranking

The discriminative model learned in the replica learning of instruction phase ranks the instances (i.e., fault localization) whether it is effective or not. The unknown instance needs to be transformed a set of feature values using the feature extraction process. These feature values are then compared with the replica and the rank is obtained. The feature vector balances with the Hyperplane that separates effective and ineffective instruction instances. The feature vector is extracted according to the side of the Hyperplane, the corresponding instance is assigned with ranking tags. FLB pseudo code is shown below

Journal of Theoretical and Applied Information Technology
20th June 2014. Vol. 64 No.2
© 2005 - 2014 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
Begin
Procedure Feature extraction with Replica Learning
Input: List of execution traces ‘P’ and elements ‘E’
Output: Discriminative Model with output score value ‘D’, ‘S’ ‘C’ and ‘B’
1: If P<E
2: Identify the similarity between the instances
   Similarity of the two vectors $\frac{\sum_{i=1}^{40}(p_i * q_i)}{\sqrt{\sum_i^{40} p_i^2 * \sum_i^{40} q_i^2}}$
// Feature Extraction
3: Let the feature extracted based on Input P1 to P5 (traces) and E1 to E4 (elements)
4: Let the features extracted be obtained through ‘E5 to ‘E10’
5: Output features capture the top 10 dishonest scores
6: Let S1- number of distinct score value; S2-Mean; S3-Median; S4- Mode; S5-Variance; S6-Standard Deviation
// Replica Learning
7: Repeat
8: Using Utmost Subsidiary Hyperplane (USH) perform replica learning
9: Until [ Hyperplane satisfies the constrains, $q_i(\vec{Z}.\vec{p}_i + c)$]
// Ranking
10: Ranks with instances whether effective or not
11: Feature Vector extracted according to side of hyperplane and accordingly assigned with tags
12: End
```

In the case of FLB pseudo code, takes as input a set of effective fault localization instances ‘P’ and ‘E’. If the program execution traces is lesser than the program elements, the feature extraction phase is performed. The replacement effective instances appear close to the Hyperplane are effectual instances while the others are the ineffective instances. In order to find these effective instances, the similarity between each effective instance is evaluated with each of the ineffective instances. Each fault localization instance viewed as a 40-dimensional cloud vector and each dimension is a feature and a localization instance is represented by the values of the 40 features.

3. EXPERIMENTAL EVALUATION SET-UP OF FLB MECHANISM

The performance of Fault Localization based on Band (FLB) mechanism is evaluated using JAVA with Hadoop. For evaluation purpose, comparison is performed on the FLB mechanism with the existing GenProg. Hadoop is an open source accomplishment of the construction for large-scale parallel data processing. Hadoop is distinction in research and data mining, so it is important to appreciate its runtime activities, pattern formation and analyze its performance. An experimental evaluation of FLB mechanism and Genprog is carried out with the Amazon EC2 dataset to estimate the performance. Amazon Elastic Compute Cloud (Amazon EC2) presents resizable calculating capability in the Amazon Web Services (AWS) cloud.

Amazon EC2 provides a broad compilation of instance types optimized on top form diverse use cases. Instance types include untrustworthy mixtures of memory, CPU, storage, and networking capability and present the litheness to decide the suitable mix of resources for the required applications. Every instance type comprises one or more example ranges, permitting to improve the resources to the supplies of the target workload. Performance metric for evaluation of FLB mechanism is measured in terms of percent time overhead, CPU utilization, performance counter, communication cost, normalized throughput, precision matchmaking, and average auditing time.

4. PERFORMANCE RESULT OF FAULT LOCALIZATION IN CLOUD ENVIRONMENT

Fault Localization based on Band (FLB) mechanism is compared against the existing Genetic Programming (GenProg) using the JAVA programming. Time overhead is the processing time required by a device (i.e.,) FLB mechanism prior to the execution of a program elements in instruction phase is measured in terms of percentage (%). CPU utilization refers to a usage of processing resources for fault localization. Actual CPU utilization in FLB varies depending on the amount and type of managed computing tasks. Certain tasks require heavy CPU time, while others require less because of non-CPU resource requirements, measured in terms of Mega Bytes (MB).

Performance counter is the effective result obtained on the overall system. The average amount of system memory used by the database manager is to hold commonly used information from the FLB mechanism to prevent file operations. Communication cost is defined as the rate it takes to identify the faults using the fault localization based on band, measured in terms of Kilo Bytes (KB).

$$W_{i,r} = \frac{instructions}{p_r}$$

Where, $W_{i,r}$ signifies the communication cost of program input trace 'i' for the program elements 'r'. p_r represent the processing capability of program elements in the FLB mechanism. Throughput is the standard rate of successful fault identification in cloud environment, measured in terms of Kilo bits per second (Kbps). Precision matchmaking in the FLB mechanism is the fraction of retrieved instances that are related to the overall system output.

$$Precision\ Matchmaking = \frac{\sum True\ Positive}{\sum Analysis\ Outcome\ positive\ value}$$

Average auditing time is the average amount of time consumed to inspect the fault from the effective and ineffective group based on the Utmost Subsidiary Hyperplane and measured in terms of seconds (sec). Table 4.1 shows the experimental values and graph illustrates the graphic form of FLB mechanism against GenProg.

Table 4.1 Tabulation of Percent Time Overhead

Sample Periods	Percent Time Overhead (%)	
	Existing Genetic Programmin g	FLB Mechanism
2	31	29
4	39	36
6	42	40
8	43	41
10	45	42
12	46	44
14	48	45
16	53	50

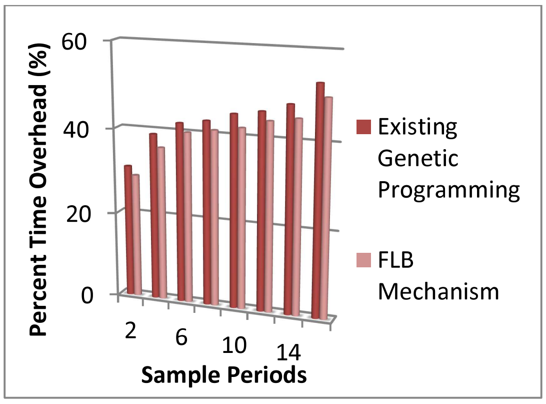

Figure 4.1 Percent Time Overhead Measure

Table 4.1 and Figure 4.1 describe the time overhead based on the sample periods observed in FLB mechanism and existing genetic programming. As the sample period increases, time overhead is reduced to 4 – 7 % in FLB mechanism when compared with the genetic programming. This is because of the reason that lesser number of trace points in cloud environment reduce the time overhead in FLB mechanism, when compared with the Genetic programming. With FLB model, the debuggers is better notified whether the output trusts of fault localization run on a set of program execution traces.

Table 4.2 Tabulation for CPU Utilization

No. of Faults Localized	CPU Utilization (MB)	
	Existing Genetic Programming	FLB Mechanism
5	15.25	14
10	16.1	14.65
15	17.5	16.21
20	18.85	16.95
25	20.28	18.85
30	21.36	19.54

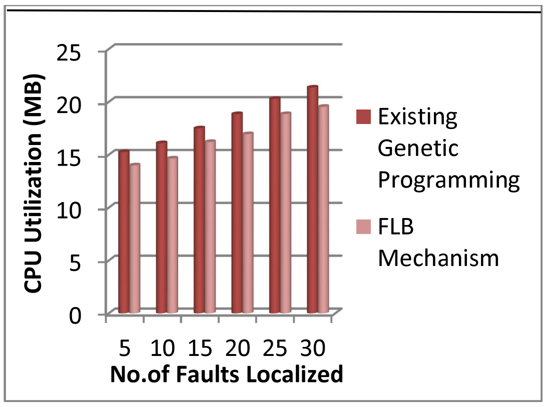

Figure 4.2 CPU Utilization Measure

Table 4.2 and Figure 4.2 describe CPU utilization of FLB mechanism and genetic programming is measured on the Amazon EC2 dataset. The CPU utilization is reduced in FLB using the machine learning framework. The set of features are appropriate for predicting the usefulness with minimal CPU resource utilization. As the fault counting ranges from 5, 10…30, CPU utilization is reduced in FLB mechanism. Utilization of CPU resources in FLB is 8 – 10 % lesser when compared with the existing genetic programming.

Table 4.3 Tabulation of Performance Counter

Problem Size (Bytes)	Performance Counter (%)	
	Existing Genetic Programming	FLB Mechanism
235	82	90
289	83	91
315	85	93
354	81	92
450	83	92
565	86	97
642	88	98

Table 4.3 describes the performance counter based on the size of information. Size is measured in terms of Kilo Bytes (KB). As the size increases, the performance count is also improved in FLB mechanism.

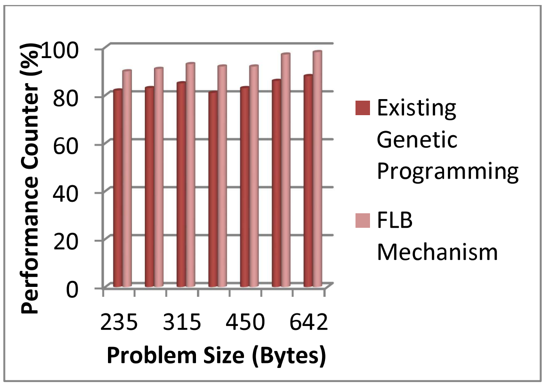

Figure 4.3 Performance Counter Measure

Figure 4.3 describes the performance counter of the FLB mechanism and genetic programming. The FLB mechanism performance result is approximately 8 – 11 % higher when compared with the genetic programming because the FLB mechanism uses band usage for fault localization that improves the performance range and the band in FLB mechanism depends on the spectra which uses the ordered list of program elements. The ordered list of program elements in

FLB mechanism are sorted based on the likelihood, to still improve the performance rate when compared with genetic programming.

Table 4.4 Tabulation of Communication Cost

Size of File (KB)	Communication Cost (KB)	
	Existing Genetic Programming	FLB Mechanism
200	262	223
400	245	210
600	366	325
800	388	338
1000	445	392
1200	555	495
1400	670	596

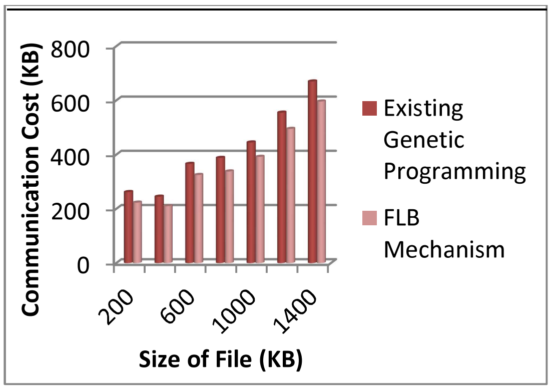

Figure 4.4 Measure of Communication Cost

Table 4.4 and Figure 4.4 describes the communication cost based on file size, whereas the file size is measured in terms of Kilo bytes (KB). The file size ranges from 200, 400, 600 up to 1400 KB. As the size increases, the communication cost incurred using FLB mechanism is reduced to 12 – 20 % when compared with the genetic programming [1]. The reduced communication cost is due to the fact that the application of Utmost Subsidiary Hyperplane separates two classes of information, resulting in the communication cost reduced in FLB.

Table 4.5 Tabulation of Normalized Throughput

No. of users	Normalized	
	Existing Genetic Programming	FLB Mechanism
3	2000	2500
6	2150	2600
9	2230	2800
12	2460	2920
15	2510	2990
18	2750	3265
21	3010	3620

Table 4.5 describes normalized throughput based on the users. At the same time, if the user count increases, throughput is improved. The normalized throughput of FLB mechanism and genetic programming is illustrated through the graph given below.

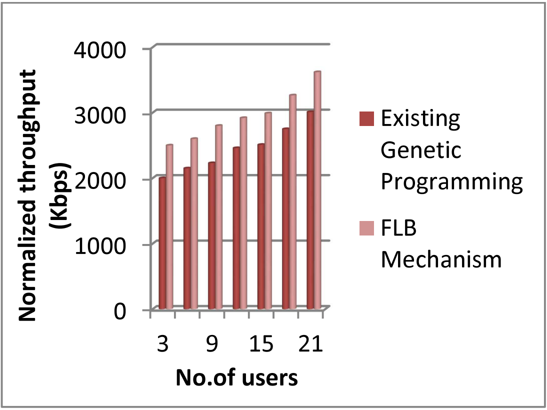

Figure 4.5 Measure of Normalized Throughput

Figure 4.5 illustrates the normalized throughput, where FLB mechanism is 15 – 22 % improved when compared with the genetic programming [1]. The FLB mechanism uses replica to convert these set of feature vectors into a discriminative model that predict the effectiveness with normalized throughput using FLB mechanism. Replica learning from instruction phase used the Amazon EC2 dataset for the evaluation of throughput.

Table 4.6 Tabulation of Precision Matchmaking

Information Size (Bytes)	Precision Matchmaking	
	Existing Genetic Programming	FLB Mechanism
33	78	90
65	79	91
94	80	92
121	82	93
156	82	96
184	83	95
215	86	97
249	87	98

Table 4.6 describes precision matchmaking effectively in FLB mechanism and genetic programming based on the information size.

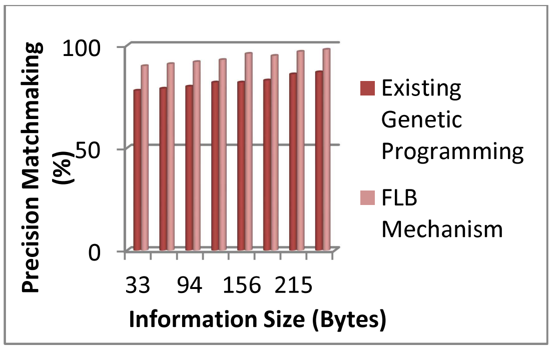

Figure 4.6 Measure of Precision Matchmaking

Figure 4.6 describes the precision matchmaking on FLB mechanism and genetic programming. As the information size varies, the precision matchmaking is 10 – 15 % improved in FLB due to the similarity between the instances is identified $\frac{\sum_{i=1}^{40}(p_i * q_i)}{\sqrt{\sum_i^{40} p_i{}^2 * \sum_i^{40} q_i{}^2}}$ using the 40 features. Each fault localization instance viewed as a 40 dimensional cloud vector matches the relevance effectively in FLB when compared with the genetic programming.

Table4.7 Tabulation of Average Auditing Time

No. of users	Average Auditing Time	
	Existing Genetic Programming	FLB Mechanism
10	119	112
20	127	122
30	149	142
40	156	148
50	167	158
60	171	163

Table 4.7 describes the average auditing time based on the users. The users count ranges from 10, 20, 30…. 70, average auditing time is reduced in FLB mechanism when compared with the genetic programming.

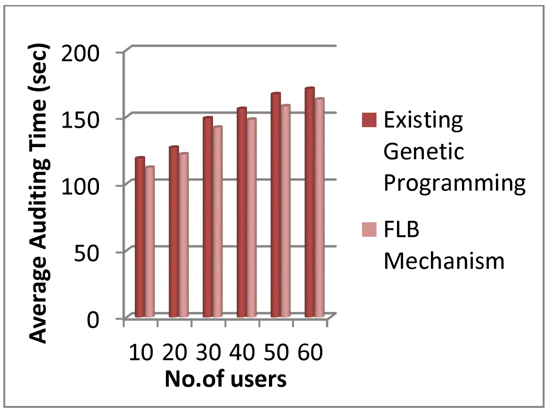

Figure 4.7 Measure of Average Auditing Time

Figure 4.7 describes the average auditing time based on the users. Top dishonest score values serve as the information summary of the score in FLB mechanism when compared with the Genetic programming. The FLB mechanism consumes 5 – 10 % lesser auditing time when compared with the Genetic programming. The break form from output features capture the dishonest scores and able to audit by localize the faults with minimal time.

Finally, it is being observed that the contributions define a new research problem by ranking the different acceptable patches. Solving the GenProg problem would help developers to better trust the output of fault localization model based on band. FLB mechanism builds machine learning process and these feature values discover out discriminative model to predict the fault localization and obtain the effectiveness in ranking.

5. CONCLUSION

Fault Localization based on Band mechanism address the faults and rank the effective group of 40 features. The values of these features from an instruction set of faulty localization build a discriminative model using machine learning. FLB extracts the features in cloud that are potentially associated for effective ranking. The FLB mechanism is then used as an ordered list of program elements sorted based on their likelihood. The techniques normally change program runtime states methodically to localize faulty program elements. FLB focus on fault localization tools that compare correct and faulty executions. The FLB machine learning process and these feature values discover out a discriminative model that predict the fault localization and effectiveness in ranking. The experimental result of FLB mechanism using Amazon EC2 dataset estimates the performance of localizing the faults. FLB attains normalized throughput, precision matchmaking, improved performance counter, approximately 5.85 % lesser percent time overhead, minimal CPU utilization, communication cost and auditing time.

REFERENCES:

[1] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest., and Westley Weimer, "GenProg: A Generic Method for Automatic Software Repair," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL.38, NO.1, JANUARY/FEBRUARY 2012

[2] David Lo., Jinyan Li., Limsoon Wong., and Siau-Cheng Khoo., "Mining Iterative Generators and Representative Rules for Software Specification Discovery," IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 23, NO. 2, FEBRUARY 2011

[3] YAN ZHU., SHANBIAO WANG., HONGXIN HU GAIL-JOON AHN., DI MA., "SECURE COLLABORATIVE INTEGRITY VERIFICATION FOR HYBRID CLOUD ENVIRONMENTS," World Scientific Publishing Company., International Journal of Cooperative Information Systems, DOI: 10.1142/S0218843012410018., Vol. 21, No. 3 (2012)

[4] Konstantinos Tsakalozos., Mema Roussopoulos., and Alex Delis., "Hint-Based Execution of Workloads in Clouds with Nefeli," IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 24, NO. 7, JULY 2013

[5] Qian Wang., Cong Wang., Kui Ren., Wenjing Lou., and Jin Li., "Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing," IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 22, NO. 5, MAY 2011

[6] Imad M. Abbadi., and Anbang Ruan., "Towards Trustworthy Resource Scheduling in Clouds," IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, VOL. 8, NO. 6, JUNE 2013

[7] Rongxing Lu., Xiaodong Lin., Xiaohui Liang., and Xuemin (Sherman) Shen., "Secure Provenance: The Essential of Bread and Butter of Data Forensics in Cloud Computing," ACM journal., 2010

[8] Balakrishnan.S., Saranya.G., Shobana.S., Karthikeyan.S., "Introducing Effective Third Party Auditing (TPA) for Data Storage Security in Cloud," IJCST Vol. 2, Issue 2, 2011

[9] Smitha Sundareswaran., Anna C. Squicciarini., and Dan Lin., "Ensuring Distributed Accountability for Data Sharing in the Cloud," IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 9, NO. 4, JULY/AUGUST 2012

[10] Yan Zhu, Hongxin Hu, Gail-Joon Ahn, Senior Member, IEEE, Mengyang Yu "Cooperative Provable Data Possession for Integrity Verification in Multi-Cloud Storage," IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, 2012

[11] Olivier Beaumont., Lionel Eyraud-Dubois., and Hejer Rejeb., "Heterogeneous Resource Allocation under Degree Constraints," IEEE

TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS., 2012

[12] Ming Li., Shucheng Yu., Yao Zheng., Kui Ren, and Wenjing Lou., "Scalable and Secure Sharing of Personal Health Records in Cloud Computing using Attribute-based Encryption," IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, 2012

[13] Cong Wang, Kui Ren, Jia Wang., and Qian Wang., "Harnessing the Cloud for Securely Outsourcing Large-scale Systems of Linear Equations," IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, 2012

[14] Vivek Nallur., Rami Bahsoon., "A Decentralized Self-Adaptation Mechanism For Service-Based Applications in The Cloud," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 2012

[15] Zhiguo Wan., June Liu., and Robert H. Deng., "HASBE: A Hierarchical Attribute-Based Solution for Flexible and Scalable Access Control in Cloud Computing," IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, VOL. 7, NO. 2, APRIL 2012 743

[16] Xuyun Zhang., Chang Liu., Surya Nepal., Suraj Pandey., Jinjun Chen., "A Privacy Leakage Upper-bound Constraint based Approach for Cost-effective Privacy Preserving of Intermediate Datasets in Cloud," IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, TPDSSI-2012

[17] Donghun Lee., Sang K. Cha., and Arthur H. Lee., "A Performance Anomaly Detection and Analysis Framework for DBMS Development," IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 24, NO. 8, AUGUST 2012

[18] Guojun Wang, Qin Liu., Jie Wu., "Hierarchical Attribute-Based Encryption for Fine-Grained Access Control in Cloud Storage Services," ACM Journal, 2010

[19] Mr. Shashikant Govind Vaidya., Prof. Mr. Shailesh Kisan Hule., Mr. Gaurav Balvant Dagade., Mr. Sharad Arjun Jadhav., "HABE (Hierarchical Attribute Based Encryption) Model for Supporting Dynamic structure of Organization," Proc. of the Second International Conference on Advances in Computing, Control and Communication (CCN)., 2012

[20] Mohamed Nabeel., Elisa Bertino., "Privacy-Preserving Fine-Grained Access Control in Public Clouds," IEEE Computer Society Technical Committee on Data Engineering, 2012