ISSN: 2278 – 909X
International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)
Volume 4, Issue 7, July 2015

System Monitoring Using Software Entropy

Mr. Rahul Lokhande
Department of Computer Science & Engineering
D. Y. Patil College of Engineering & Technology, Kasaba Bawda, Kolhapur
Maharashtra, India

Abstract- ***Many techniques have been proposed to execute this kind of monitoring but failed to provide a feasible solution. The aim of this paper is to propose a solution of system monitoring with the help of Information Theoretical approach involving software metrics and their respective entropies. This paper also provides the fault detection and fault diagnosis techniques to locate the faults within the clusters of the software metrics and their entropies. Software entropy is considered as a measure to detect faults. Information theoretical approach is used in this paper. In this paper the variation of the K-means Algorithm is used for the purpose of the clustering, for the fault detection Wilcoxon Rank sum Test is used which is nonparametric in nature, for the diagnosis purpose three diagnosis algorithms are used namely, Ratio Score, Sigscore and Bayesian score. The result of the experimentation clearly shows the better system monitoring option.***

Keywords- *software metrics, entropy, mutual information, clusters, fault detection, fault diagnosis etc.*

1. INTRODUCTION

1.1 Overview

Software systems continue to grow in size and complexity as more functionality is implemented . Software systems usually start small and simple, but in less period of time they becomes complex because of more number of features and requirements are added into the system. As more number of components added the interactions of system grows in non-linear fashion. In mid October 2001, IBM released declaration policies, which contained the observations that the main obstacle for further progress in the IT industry is dominating software complexity crisis. The observations also suggested that the difficulty of managing today's computing systems goes well beyond the administration of individual software environments. Computing System complexity appears to be approaching the limits of human capabilities. This system approach to characterizing the normal behavior of a system entails finding and modeling stable relationships between system metrics. Software faults are difficult to remove completely from the system, so it is unrealistic to expect a complex software system is fault free. The major challenge for the system management is to detect and isolate the software faults effectively from complex software systems. In this paper, we take an information-theoretic approach

to capture inter-metric relationships without the need to commit to any specific mathematical form. Further, rather than considering metric pairs, we group similar metrics and monitor the resulting clusters, yielding significant efficiency gains. We monitor the system by tracking in-cluster entropy for each cluster. A large amount of monitoring data can be collected from the system components for fault analysis. This data can be used for the purpose of monitoring and also for the fault detection and diagnosis. The system monitoring with fault detection and diagnosis consists of

different phases to properly analyze collection of metrics in fault detection and diagnosis. he base for the consideration of this project is the need for genuine system monitoring which gives the fault detection and the fault diagnosis as a composite function. The system monitoring implemented in this project is not used collectively in previous techniques. Previous techniques gives the whole different approaches for the fault detection and the diagnosis, which require more synchronization between the two techniques used, because these two were not come under the common approach as in this project. This project will serve as a more useful function which is build on the basic properties of the project and undoubtedly clear parameters such as software metrics. With the help of the software entropy values are being calculated very carefully so, it can be used for further tasks in system monitoring. The fault detection gives non-parameterized technique of detection. Along with this the diagnosis gives the diagnosis observations based on the three different diagnosis algorithms. This project solves the problem of system monitoring, collectively with fault detection and the diagnosis in a single project, which is an uncommon feature of the system monitoring. It also provides parameters which have clear values i.e. software metrics and use of the software entropy gives the monitoring parameters to be observed. It fulfills the need of the monitoring of the software for the fault detection in the normal execution.

2. RELATED WORK

2.1 Background

There is a considerable work in the field of the identifying the relationship between the metrics. In the well- behaved system the stable relationships between the metrics are exist [1], [2], [3]. When error occurs in the relationship between the metrics are distributed. Different, modeling techniques have been proposed to represent the relationship between the metrics in the system. These techniques are Simple Linear Regression [3], Simple Linear Regression with Transformed data (SLRT) [11], Locally Weighted Regression (LWR), Autoregressive regression with eXogenous Input (ARX); [13] Entropy based modeling,[14], Gaussian Mixture models [1]. These systems have certain limitations with them. The prior techniques assume a specific mathematical from of the relationship between the metrics, but it is not guaranteed that there present a specific mathematical form for the relationship between the metrics. Techniques like Simple Linear Regression (SLR) are not able to capture the non-linear dynamics, so that they can miss the number of the stable relationships. On the other hand, the techniques like Gaussian Mixture Model (GMM) and Locally Weighted Regression (LWR), which deal with the non-linear dynamics, are very costly and also they require very careful parameterization which can result into the difficult general applications. The approach of combining the different techniques also increase the computational overhead as well as they requires separate modeling technique and parameter estimation. This shows that the previous work in this field is on the specific mathematical form and they are prohibitive linearly computationally.

The current techniques available for the Fault diagnosis of the system based on the metric correlations are variants based on Jaccard coefficient. These techniques are evaluated related to the metric-pair methods. One of the techniques used for diagnosis is Pinpoint which is problem determination in large dynamic internet services. This technique makes use of the user requests tracing for the diagnosis, which is more expensive. There have been very few studies on software fault prediction without prior fault data. Zhong et al. proposed a clustering and expert-based software fault prediction method when the fault labels for software modules are unavailable. They used K-means and Neural- Gas clustering algorithms to cluster modules. After the clustering phase, a 15-years experienced software engineering expert examined the representative

module (mean of each metric) of each cluster and several statistical data such as maximum, minimum, median of each metric, in order to label each cluster as fault-prone or not. AMPLE (Analyzing Method Patterns to Locate Errors) is a system for identifying faulty classes in object-oriented software. It collects hit spectra of method call sequences, which are subsequences of a given length that occur in a full trace of incoming or outgoing method calls, received or issued by individual objects of a class. Each call sequence is assigned a weight, which captures the extent towhich its occurrence or absence correlates with the detection of an error, i.e., it is a combined measure of similarity call sequences of a class, leading to a class weight. Classes with a high weight are most likely to contain the fault that causes the detected error. Although the calculation of the sequence weights in AMPLE can be explained as an application of the technique of diagnosis is at class level, and the calculated coefficients are used only to collect evidence about classes, not to identify suspicious method call sequences. Concluding, we can observe that three existing tools for diagnosis and automated debugging rely on an analysis of program spectra. Program spectra themselves were introduced in [25], where hit spectra of intra-procedural paths are analyzed to diagnose year 2000 problems. In the distinction between count spectra and hit spectra several kinds of program spectra are evaluated in the context of regression testing. As it already mentioned in the introduction, in the context of computer programs, fault localization based on the analysis of program spectra is an automated debugging technique. An example of a different (black box) technique in that category is Delta Debugging [26], which compares the program states of a failing and a passing run, and actively searches for failure-inducing circumstances in the differences between these states.

2.2 Need of the Work

Prior researches in this field pointing the stable relationships between the metrics exist in the well-behaved system, but again the relationships gets disturb with the occurrence of the error, so these methods are quite unreliable. Also the prior techniques assume to have a specific mathematical form of relationship between the metrics, which is hard to accept and the assumption does not work every time.

The previous techniques were proposed the relationships between the metrics, but they consider only the metric pair, which makes the job more tedious as each time to pair and examine the metrics takes a long time. Other non-linear techniques are costly and their requirement of the careful parameterization makes it difficult to work with general applications.

The need arises for the different system monitoring, which provides the relationship between the metrics without considering any specific mathematical form. The system considers the similar metrics together as a cluster. This system applies the techniques on the cluster for fault detection. Also the method should provide the detection of the fault automatically with the help of ranks of the clusters and diagnose the faults with the help of various diagnosis algorithms.

3. SOFTWARE METRICS

3.1 Software metrics

3.1.1 Introduction

Software is the collection of computer programs, procedures, rules, associated documentation and data which are collected for specific purpose. Software is the various kinds of programs used to operate computers and related devices. A program is a sequence of instructions that tells a computer what operations to perform. Programs can be built into the hardware itself, or they may exist independently in a form known as software. According to the software engineering context, a measure provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of

a product or process. Measurement is the act of determining a measure.

The software crisis must be addressed and possibly resolved. For accomplishing this task it requires more accurate schedule and cost estimates, better quality products and higher productivity. All these can be achieved through more effective software metric use. The *IEEE Standard Glossary of Software Engineering Terms* defines *metric* as "a quantitative measure of the degree to which a system, component, or process possesses a given attribute."

Metrics should facilitate the development of models that are capable of predicting process or product parameters; they do not have to just describe them. The good metrics should posses following properties:

- *Simple* - Software metric should be simple so it should be clear how to evaluate them.
- *Objective*- Software metric should have the objective to the greatest extent
- *Easily Obtainable*- The metrics should be easily obtainable in a reasonable cost.
- *Valid*- The metric should measure what it is intended to measure.
- *Robust*- Metric should be relatively insensitive to insignificant changes in the process.

3.1.2 Classification of the Metrics

Software metrics can be classified as either product metrics or process metrics.

Product Metrics- measures of the software product at any stage of its development from requirements to its installed system. Product metrics may measure the complexity of the software design, the size of the final program, or number of pages of documentation produced.

Process Metrics- measures of the software development process, such as overall development time, type of methodology used, or the average level of experience of the programming staff.

Other way to classify the software metrics are: objective and subjective

Objective Metrics- these metrics always results in identical values for given metric as measured by two or more qualified observers.

Subjective Metrics- even qualified observes might measures different values for a given metrics.

For product metrics, size of the products measured in lines of code (LOC) is an objective measure for which any informed observer should obtain the same measured value for a given program. For process metrics, development time is the objective measure and level of programmer experience is the subjective measure.

Another way to classify the software metric: Primitive and computed metrics

Primitive Metrics- the metrics that can be directly observed, for example program size (in LOC), number of defects observed in unit testing, or total development time for the project.

Computed Metrics- The metrics which cannot be directly observed, but they are computed in some manner from the other metrics. Examples are the metrics which are commonly used for the productivity, LOC produced per person – month or for product quality such as the number of defects per thousand lines of code.

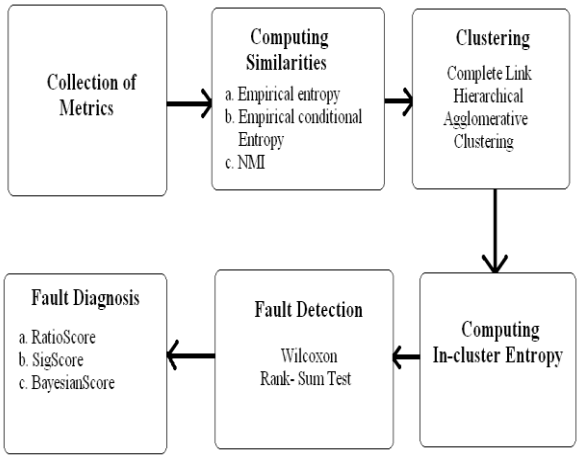

Figure-Architecture of the work

3.1.3 Software Metric Examples

Following are some examples of the software metrics:

- Number of packages
- Number of methods

- Lines of code (LOC)
- Number of Interfaces
- NSM- Number of static methods
- TLOC- Total Lines of codes.
- CA- Afferent Coupling
- RMD- Normalized Distance
- NOC- Number of Classes
- SIX- Specialization Index
- NOF- Number of Attributes
- NOP- Number of Packages
- WMC- Weighted Method per Class
- NSF- No. of static Attributes
- NORM- Number of Overridden Methods
- NOM- Number of Methods

These are few examples of software metrics which represents the state of the software from which they are collected. A right set of metrics can provides the idea about the working state of the software as well as the performance of software.

3.1.4 Software metrics used in the project

For this project work this project used the Chidamber & Kemerer Java metrics, which is which is an open source command line tool. It calculates the C & K Object Oriented metrics by processing the byte-code compiled Java files. Following are the list of software metrics used in this project.

- WMC- weighed methods per class
- DIT- Depth of Inheritance Tree
- NOC- Number of Children
- CBO- Coupling between object classes
- RFC- Response for a class
- Ca-Afferent Coupling
- NPM- Number of Public Methods
- Ce- Efferent Coupling
- DAC- Data Abstract Coupling
- MPC- Message Passing Coupling
- COF- Coupling Factor
- MIC- Method Invocation
- ICP- Information flow based coupling

4. ENTROPY OF SOFTWARE METRICS

4.1 Software Entropy

The second law of thermodynamics, in principle, states that a closed system's disorder cannot be reduced; it can only remain unchanged or increase. A measure of this disorder is entropy. This law also seems plausible for software systems; as a system is modified, its disorder, or entropy, always increases. This is known as software entropy. The inspiration for adopting the word *entropy* in information theory came from the close resemblance between Shannon's formula and very similar known formulae from thermodynamics. In statistical thermodynamics the most general formula for the thermodynamic entropy *S* of a thermodynamic system is the Gibbs entropy.

Information theory is a branch of applied mathematics, electrical engineering, bioinformatics, and computer science involving the quantification of information. Information theory was developed by Claude E. Shannon. Entropy is known as the key feature of the information. Entropy quantifies the uncertainty involved in predicting the value of a random variable. Information entropy measures the uncertainty or unpredictability of the random variable. For discrete random variable X, the entropy is given by,

$$H(X) = E_p \log\frac{1}{p(X)} = -\sum_{i=1}^{n} p(x_i)\log p(x_i)$$

where X takes values from the set $\{x_1; x_2; \ldots; x_n\}$; Ep refers to the expectation with respect to the probability distribution of X characterized by the density function p. If $p(X = x_i) = 1$ and $p(X = x_j) = 0$ for any $i \neq j$, i.e., there is no uncertainty about X, then $H(X)$ is zero. Otherwise, $H(X)$ takes a positive value.

4.2 Conditional Entropy

The conditional entropy measures the uncertainty of a random variable Y given another random variable X. It represents the remaining uncertainty of Y knowing values taken by X. It is defined by

$$H(Y|X) = E_p \log \frac{1}{p(Y|X)}$$

$$= -\sum_{i=1}^{n}\sum_{j=1}^{m} p(x_i, y_j) \log p(y_j|x_i)$$

4.3 Mutual Information

The mutual information measures the reduction in uncertainty of a random variable Y given another random variable X. This reduction represents the amount of information either variable provides about the other. It is defined by

$$I(X,Y) = H(Y) - H(Y|X)$$

4.4 Normalized Mutual Information

However, it is impractical to use either conditional entropy or MI as a measure of the similarity between *X* and *Y*. Conditional entropy is not symmetric, i.e., *H (Y |X)* is often not equal to *H (X|Y)*. While MI is symmetric, its absolute value is not necessarily comparable across random variables. MI is influenced by *H(X)* and *H(Y)*, which may have different maximal values. Strehl and Ghosh [7] developed normalization for mutual information, called Normalized Mutual Information (NMI), to address the shortcomings
of MI. It is defined by

$$NMI(X,Y) = \frac{I(X,Y)}{\sqrt{H(X)H(Y)}}$$

For any random variables, *X* and *Y*, NMI has following properties

1. $0 \leq NMI(X, Y) \leq 1$.
2. $NMI(X, Y) = NMI(Y, X)$.
3. If X and Y are independent, NMI(X, Y) = 0.
4. If Y = f(X), NMI(X, Y) = 1, for any invertible function f.

The more correlated two variables are, the higher NMI they have, in spite of the specific form of their relationship. Because of these properties NMI provides a good measure of the correlation between two variables, and can be used as a similarity measure.

4.5 Empirical Entropy, Empirical Conditional Entropy and Similarity Matrix

The metric values are periodically been collected for calculating the Empirical Entropy, after collecting the values following formula is applied on the random variable X

$$H(X) = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}$$

The Empirical Conditional Entropy can also be calculated using the formula:

$$H(X) = -\sum_{i}\sum_{j} \frac{n_{ij}}{n} \log \frac{n_{ij}}{n_i}$$

Where n_{ij} is the number of samples (x, y) with x in bin i and y in bin j.

With the help of Empirical Entropy and Empirical Conditional Entropy the NMI will be calculated, which will form a metrics Similarity Matrix.

5. CLUSTERING AND CLUSTER ENTROPY

5.1 Clustering

If both (X, Y) and (Y, Z) are strongly correlated, then (X, Z) are also likely correlated. This implies that there exist groups of mutually correlated metrics, or clusters. This is used to group the metrics having the similar values. This is done by using the Complete Link Hierarchical Clustering algorithm. By getting the

similarity matrix from the complete link hierarchical agglomerative clustering is applied to group the similar metrics. This algorithm works by grouping the metric one by one on the basis of nearest distance measures of all the pair wise distance between the metrics.

Algorithm for clustering:

Steps:

1. Take similarity matrix M as a input
2. Define the threshold for maximum distance between two clusters.
3. Define the maximum distance between the two clusters as the distance between the two clusters.
4. Treat every metric as a single cluster and merge nearest cluster until either the distances between every two clusters exceed the predefined threshold or all metrics belong to one cluster.

The algorithm gives the guarantee about all metrics in cluster have similarity of the predefined threshold. The algorithm gives facility that the specification of clusters a priori is not necessary.

5.2 Cluster entropy

After forming the clusters of the software metrics having the similar values with the help of the Similarity Matrix, the next step will be computing the entropy of the clusters created. This module computes the status of the each cluster in the form of entropy. The metrics in one cluster are closely correlated with each other since, the relationships between the metrics are either linear or nonlinear it is difficult to establish analytical models for the cluster or for the metrics in the cluster.

The empirical entropy can be taken as the signature of the cluster, which provides the status of the cluster. For a given cluster, a significant change in the behavior of the cluster entropy indicates a fault.

The steps for computing cluster entropy:

1. Normalization of all metric values by dividing each value with its average values which is based on data collected during normal operations.
2. Relate the different metric values to a single variable.
3. Calculate the empirical entropy of the random variable over time.
    1. Set the range to[0,7]
    2. Divide it into seven equal bins.
    3. Add an eighth bin with the range [7, ∞], for the values that do not fit in the other seven bins.
    4. Monitor the cluster entropy over time i.e. tracking the entropy of each cluster.

6. FAULT DETECTION AND DIAGNOSIS

6.1 Fault Detection

Consider the entropy behavior of the clusters. They show the behavior of cluster entropy when some fault occurs at time. Human operators can readily identify unusual changes in the pattern of the in-cluster entropy, and, as a result, suspect errors. It is impractical to have these operators continuously track the behavior of all clusters. Instead, automatically identifying anomalies in the in-cluster entropy is nontrivial because there are no general rules that differentiate between normal and disturbed behavior.

For this purpose nonparametric statistical test namely Wilcoxon Rank-Sum Test is used to identify significant shifts in In-cluster entropy of individual clusters for detecting the fault within the systems.

Let E_i be the In-cluster entropy of cluster E at time i. For detecting the significant change in Ei when fault occurs, two sliding windows of Ei are kept. The test window consists of most recent nEi's and the baseline window consists of mEi's before the test window, then applying the Wilcoxon Rank-Sum Test on these windows. If the significant change is indicated by the test alarm is raised.

For example, if $\{E_{s+1}, E_{s+2}, \ldots, E_{s+m}\}$ and $\{E_{s+m+1}, E_{s+m+2}, \ldots, E_{s+m+n}\}$ are two sample sets from two sample windows $(s+1, s+m)$ and $(s+m+1, s+m+n)$ are from the same distribution, then Wilcoxon Rank-Sum statistic is given by the formula:

$$W = \sum_{i=1}^{m}\sum_{j=1}^{n} h_{s+i,s+m+j} + \frac{m(m+1)}{2}$$

Where

$$h_{i,j} = \begin{cases} 1, & X_i < X_j \\ 0.5, & Xi = X_j, \\ 0, & \text{otherwise.} \end{cases}$$

6.2 Fault Diagnosis

The goal of diagnosis is to determine the cause of errors (i.e., to localize the fault). The faster the source of an error can be found, the faster its cause can be addressed. This
reduces the amount of downtime the system incurs, thereby improving system availability.

In this module, the diagnosis algorithms are used to locate the faults correctly. A system *S* is the collection of the subsystems *S1, S2, S3…, Sn*. Set of all metrics be M a set of all clusters be *C*. There is n-to-1 mapping between metrics to subsystems $\alpha: M \quad S$, the set of all the *n*-to-1 mappings α is called as *A*. There is *n*-to-1 mapping between metrics to clusters $\beta: M \quad C$, the set of all the *n*-to-1 mappings β is called as *B*. At any time *t*, each cluster can be checked for the anomalies. Record will be kept as $o(t) = 1$, if *Ci* reports anomaly, $o(t) = 0$ otherwise. This results into an Observation vector $O(t) = [o_1(t), o_2(t), \ldots, o_m(t)]^T$.

Inputs for the diagnosis algorithms are the matrix *M* and the observation vector *O(t)* which gives the output as anomaly score vector $r = [c_1, c_2, \ldots, c_m]^T$ such that the subsystem *Si* is considered as more faulty than the *Sj* if $c_i > c_j$..

Following diagnosis algorithms are used:

1. RatioScore
2. SigScore
3. BayesianScore

6.2.1. RatioScore

RatioScore entails counting the number of times a component is found in anomalous clusters. The idea behind RatioScore is that a faulty component is likely to cause the clusters which contain the component's metrics to show anomalous behavior. As a result, a component has a higher anomaly score than other components if more clusters
containing metrics of those component exhibit anomalies.

Inputs for the RatioScore are model subsystem association matrix *M*, and observation vector *O (t)*, using Jaccard coefficient for the assignment of the anomaly scores to each component:

$$r_j = \frac{\sum_{i=1}^{m} c_i(t) \cap M_{ij}}{\sum_{i=1}^{m} c_i \cup M_{ij}}$$

6.2.2 SigScore

This algorithm is based on the observation that the anomaly score of a component on which many others depend (i.e., the more popular or significant it is) tends to be too high and often less reliable. For adjusting the score got by the RatioScore, the SigScore is calculated by using the formula:

$$r_j = \frac{\sum_{i=1}^{m} c_i(t) \cap M_{ij}}{\pi(C_j)\sum_{i=1}^{m} c_i(t) \cup M_{ij}}$$

where, $\prod (Cj)$ is a probability distributed vector.

6.2.3 BayesianScore

SigScore takes the relative significance of components into account and adjusts the RatioScore in an efficient way, it is not well-justified theoretically. Bayesian inference rules are used to infer the likelihood of each

component being faulty. Bayesian inference is a well established method to infer the cause given results. In the system when fault is the cause and disturbed correlations are the results; therefore, Bayesian inference is a good algorithm for diagnosis. Bayesian inference rules to infer the likelihood of the components being faulty. *Sj* denote the event that the fault is in component *j*. The probability of *Sj*, given the observation vector *O* is given by:

$$P(S_j|O) = \frac{P(O|S_j)P(S_j)}{P(O)}$$

$$= \frac{P(O|S_j)P(S_j)}{\sum_{j=1}^{n} P(Sj)P(O|S_j)}$$

P (O|Sj) can be computed using frequently used Bayesian assumption:

$$P(O|S_j) = \prod_{i=1}^{m} P(O_i|S_j)$$

Three cases are considered for reporting the cluster faulty

1. False alarm
2. Error spread
3. Contains metrics from faulty components

Notations according to the cases mentioned above

1. β_1 for false alarms
2. *Cj* reports an error because of error spread by the *n(Cj)*
3. β_2 indicates metrics from the faulty component

Estimation of *P (Oi|Sj)* can be done as follows:

$$P(O_i = 1|S_j) = \beta_1 + (1-\beta_1)(\pi(C_j) + (1-\pi(C_j))M_{ij}\beta_2)$$

$$P(O_i = O|S_j) = 1 - P(O_i = 1|S_j)$$

7. EXPERIMENTAL RESULTS

For experiments, JAVA source codes are used. Following are the results obtained and its analysis

7.1.1 Cluster size Vs Fault Detected

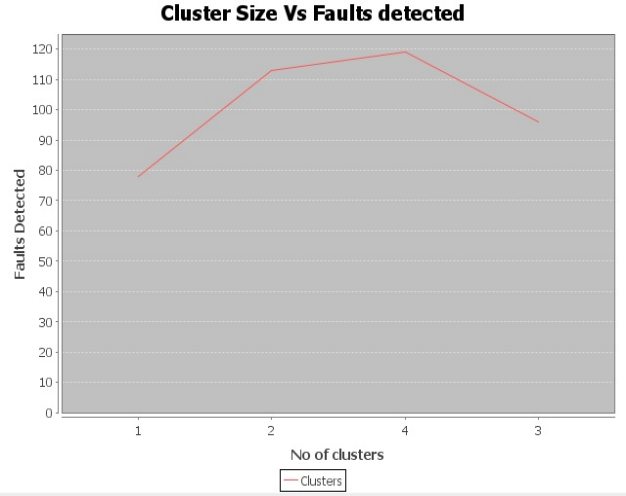

Figure 7.1- Graph showing relationship between Cluster Size & Faults Detected.

In this graph the relationship between the cluster size and the faults detected during the implementation of the work is represented. The graph values are collected by running the project on the various numbers of source codes. The graph shows the gradual increase as the size of the clusters gets increased. The reason for the increment of faults detected is, as the clusters size increased the different values of the software metrics are observed they results into the increment in the faults detection. More number of software entropy values will give increased number of the In-cluster entropies which again results into the increased fault detection. The graph suggests that as more number of the clusters involved the complexity increased as a result the flow gets disturbed which results into possible increase in faults detection.

7.1.2 NMI VS Faults detected.

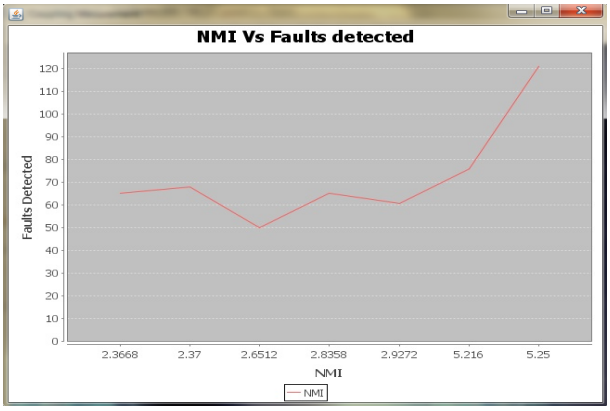

Figure 7.2 - Graph showing relationship between NMI Threshold & Faults Detected.

The graph represents the relationship between the NMI (i.e. Normalized Mutual Information) threshold and the faults detected. The graph values are observed on the basis of various number of source code running on the project. For each project the values collected of the NMI are more in number with comparison of other collected values. The increment in the NMI threshold shows the increased difference in the metric entropies which gives large difference in the cluster entropies which results into the increased fault detection. On the basis of the graph, evaluation can be made as the NMI is directly proportional to the faults detected.

7.1.3 Candidate Set-Size VS Fault coverage

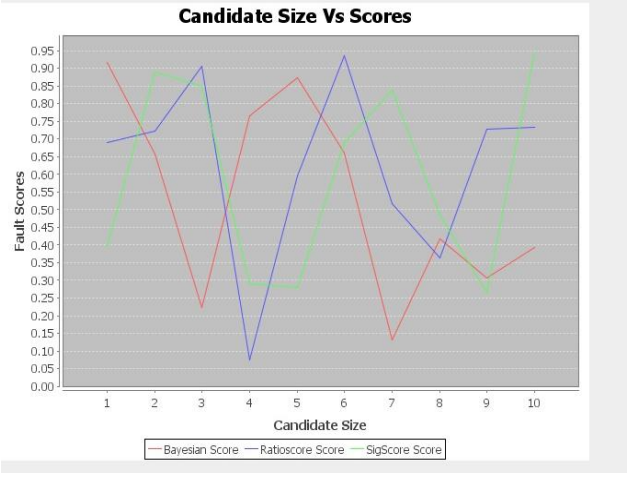

Figure 7.3 - Graph showing relationship between Candidate set size & Fault Coverage.

The graph shows the relationship between the candidate set size and the faults coverage by the diagnosis algorithms Ratio Score, SigScore and Bayesian Score used. The graph values are collected from running the various source codes on the project. The values of the diagnosis algorithms are collected with respect with the different cluster sizes. The observation of the graph is that the Ratio Score diagnosis algorithm gives the primary diagnosis of the faults whose values are changed randomly. The sigscore suggests the correction of the diagnosis of the ratio score and also gives the nearby values which are results of the ratio score. The Bayesian score line in the graph shows a little randomly behavior with the increased candidate set size. The reason for the graph line behavior is increased candidate se size may increase the faults detected which ultimately results into the diagnosis behavior observed with the diagnosis algorithms. Bayesian score is represented with the blue line, sigscore is presented with the light green line and the Ratio Score is shown with the red line.

8. CONCLUSION

In this project, n approach is presented which is built on information theoretic measures, to automatically monitor the health of complex software systems and localize faulty components when faults occur. The approach consists of tracking the entropy of metric clusters. I employ the Wilcoxon Rank-Sum test to automatically identify significant changes in cluster entropy, thereby enabling robust fault detection. For diagnosing faulty components, I extend the use of the Jaccard coefficient to clusters of metrics. In addition, I present SigScore and BayesianScore, two new diagnosis algorithms motivated by PageRank. I evaluate the approach using a various software systems. I show through experiments that the fault-detection approach has high fault coverage I also show improvement in diagnosis when using the SigScore and BayesianScore algorithms. Results indicate that the proposed diagnosis algorithms can provide valuable help for addressing faults in complex systems.

FUTURE WORK

The SigScore and BayesianScore algorithms require information on component dependencies. However, complete dependency information is not always available. Also,
in dynamic systems, dependencies may change. To study the effect of incomplete and dynamic dependency information on diagnosis is part of future work.

REFERENCES

[1] Z.Guo, G. Jiang and K.Yoshihira, "Tracking Probabilistic Correlation of Monitoring Data for Fault Detection in Complex Systems", *Proc int'l Conf. Dependable Systems and Networks (DSN '06). 2006.*

[2] G. Jiang. H. Chen and K.Yoshihira, "Modeling and Tracking of TransactionFlow Dynamics for Fault Detection in Complex Systems" *IEEE Transactions Dependable and Secure Computing, Oct-Dec 2006.*

[3] M. A. Munawar and P. A. Ward "Adaptive Monitoring in Enterprise Software Systems" *Proc .Workshop Systems Modeling Language (SysML), June2006*

[4] J. O. Kephart and D.M. Chess, "The Vision of Autonomic Computing" *Computer, vol.36, Jan 2003.*

[5] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox and E.A. Brewer, "PinPoint: Problem Determination in Large Dynamic Internet Services" *Proc int'l Conf. Dependable Systems and Networks (DSN). 2002.*

[6] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox **"**Capturing, Indexing, Clustering and Retrieving System History" *Proc.Symp. Operating Systems Principles (SOSP), 2005*

[7] "An Evaluation of Similarity Coefficients for Software Fault Localization", Rui Abreu, Peter Zoeteweij, Arjan J.C. van Gemund Software Technology Department Faculty of Electrical Engineering, Mathematics, and Computer Science Delft University of Technology.

[8] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic internet services. In Proceedings of the 2002 International Conference on Dependable Systems and Networks, pages 595– 604, Washington, DC, USA, 2002. IEEE Computer Society.

[9] "Scenarios Where Utilizing a Spline Model in Developing a Regression Model is Appropriate", Ning Huang, University of Southern California.

[10] "Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting", William S. Cleveland; Susan J. Devlin, Journal of the American Statistical Association, Vol. 83, No. 403. (Sep., 1988), pp. 596- 610.

[11] "Parameterization of multi-output autoregressive-regressive models for self-tuning control", Mirosla Vkárný.

[12] "Adaptive Monitoring with Dynamic Differential Tracing- Based Diagnosis", Mohammad A. Munawar, Thomas Reidemeister, Miao Jiang, Allen George, and Paul A.S. Ward.

[13] "A Review of Fault Detection Techniques to Detect Faults and Improve the Reliability in Web Applications", Jyoti Tamak**,** Department of Computer Science and Engineering, University Institute of Engineering & Technology Kurukshetra University, Kurukshetra, Haryana, India.