Using Hessian Locally Linear Embedding for autonomic failure prediction

Xu Lu, Huiqiang Wang, Renjie Zhou, Baoyu Ge
College of Computer Science and Technology, Harbin Engineer University,
Harbin 150001, China
{luxu, wanghuiqiang, zhourenjie,gebaoyu}@hrbeu.edu.cn

***Abstract*—The increasing complexity of modern distributed systems makes conventional fault tolerance and recovery prohibitively expensive. One of the promising approaches is online failure prediction. However, the process of feature extraction depends on the experienced administrators and their domain knowledge to filtering and compressing error events into a form that is easy for failure prediction. In this paper, we present a novel performance-centric approach to automate failure prediction with Manifold Learning techniques. More specifically, we focus on methods that use Supervised Hessian Locally Embedding algorithm to achieve autonomic failure prediction. In our experimental work we found that our method can automatically predict more than 60% of the CPU and memory failures, and around 70% of the network failure based on the runtime monitoring of the performance metrics.**

Keywords-failure prediction; autonomic computing; manifold learning; Hessian Locally Linear Embedding

I. INTRODUCTION

Failure prediction for large scale distributed systems is crucial towards further enhancement of system dependability [1].The ability to analyze data in real time and to predict potential failure is also regarded as the core of autonomic computing [2]. By runtime monitoring the system current state and then performing online failure predicting, administrators can apply preventive maintenance measures to mitigate or prevent failure occurrence. Therefore, Anticipating failure before applying preventive strategies to avoid failure occurrence is in an urgent need.

However, failure prediction has long been considered as a daunting challenge mainly due to the lack of in-depth understanding of failures and their empirical and statistical properties. Recently there have been some attempts to build prediction models or heuristic rules by discovering failure patterns for large scale systems, such as IBM BlueGene/L supercomputers [3, 4]. To achieve efficient failure prediction, a set of raw features which can accurately capture the characteristics of failures need to be extracted from the RAS (reliability, availability, serviceability) event logs. Though these methods work very well on their own systems, there are still problems which limit their popularization on other distributed systems. Firstly, the raw event logs often contain an enormous amount of entries, many of which are repeated and redundant. Before using these logs to study the failure behavior, system administrators must first filter the failure data and isolate unique failure, which has been shown a challenging task [5]. Secondly, to make raw feature sets suitable for prediction, it is necessary to identify the features from events, including normalizing the numeric feature values and calculating each feature's significance major [4].

To address above issues, in this paper we utilize a Manifold Learning (ML) technique to achieve autonomic failure prediction in distributed systems. The basic idea of our approach is to predict failures by recognizing patterns of performance metrics that indicate an imminent failure. In our previous experiment, we used locally linear embedding algorithm to extract failure features. However, the classification results were unstable and the prediction accuracy was low. Therefore, in this paper we propose a novel approach based on Supervised Hessian Locally Linear Embedding (SHLLE) algorithm to automatically extract the failure features, and show that it can achieve effective online failure prediction. We compared the various performance metrics under faulty and non-faulty conditions after feature extraction. Thus we can easily recognize the failure-prone patterns and then apply some preventive measures to prevent failure.

II. RELATED WORK

Though it is widely considered that failures are difficult to predict, it is worthwhile to revisit the feasibility of predicting failure occurrences. Regardless of a lot of works providing methods developed in combination with reliability theory, there were also a variety of runtime monitoring based methods for short term failure prediction. For example, some approaches are built on failures that have occurred during runtime rather than previous failure rates record [6, 7]. More prediction techniques are based on the faults symptom monitoring. These methods periodically evaluate system performance metrics in order to estimate a trend of deviation from normal system behavior [8,9,10].

As the number of ways a system could fail continued to grow rapidly, the type and sequence of errors are used to prediction failures. This is reflected in recently developed prediction approaches. For example, an online prediction approach was introduced to forecast the occurrence of failures by recognizing failure-prone patterns of error events [11]. Another well known prediction algorithm has been developed by applying data mining techniques to identify sets of events that are indicative of the failure occurrence [12]. More recently, various prediction methods for IBM Blue Gene/L system has been developed [1,3,4]. The RAS event log produced by an IBM Blue Gene/L system was used to find patterns relating non-fatal and fatal events. By addressing the issue of failure prediction as a classification problem, an event prediction was made when a set of learned patterns against the data matched. While these methods are effective, the cost of application on other distributed systems is high due to the implementation of RAS log collection and the requirement of expert's domain knowledge.

III. OVERVIEW

A. *Framework*

As shown in Figure 1, an autonomic failure prediction framework typically consists of three functions: performance monitoring, feature extraction, and failure prediction. A prediction process may run like the following three steps: Performance monitoring collects data of various performance metrics under faulty and non-faulty conditions. Then, feature extraction performs manifold learning techniques to deduct dimensionality and obtain the extracted features. By comparing the runtime trends in the behavior of performance metrics with the failure samples, the corresponding classifier will provide failure alarm to guide proactive recovery operations. Correspondingly there are two phases in failure prediction: training phase and testing phase. In the phase of training, we adapted SHLLE algorithm to reduce the dimensionality of samples to optimize the SHLLE parameters setting. While in the test experiment due to the computational cost of the nonlinear mapping, we use nonparametric generalization to approximate such mapping.

It is worth mentioning that the point of this paper is not to showcase either the dimensionality deduction algorithm or the failure pattern recognition heuristics. Instead, the idea is to study the usage of manifold learning techniques to failure prediction and to evaluate the feasibility of its autonomic prediction strategy driven by high-dimensionality samples.

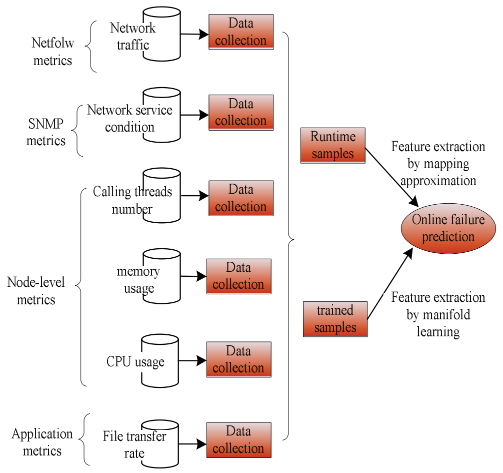

Figure 1. ML-based failure prediction framework

B. *Supervised hessian locally linear embedding*

As stated in previous section, dimensionality reduction is an important step before classifying multidimensional data for failure prediction. Hessian Locally linear embedding is one of the methods intended for this task [13]. The Hessian locally linear embedding algorithm mainly contains following steps:

1. Identify neighbors: for each data point m_i , i =1...n, identify the indices corresponding to the k-nearest neighbors in Euclidean distance, let Γ_i denote the collection of those neighbors. For each neighborhood Γ_i , i =1...n, form a k×n matrix M^i with rows that consist of the re-centered points $m_j - \overline{m_i}, j \in \Gamma_i$ where $\overline{m_i} = Ave\{m_j : j \in \Gamma_i\}$.

2. Obtain tangent coordinates: Perform a singular value decomposition of mi and getting matrices U, D and V, U is k by min (k, n), the first d column of U give the tangent coordinate of point in Γ_i .

3. Develop Hessian estimator: Develop the infrastructure for least-squares estimation of the Hessian. The matrix H^i hold the property that if f is a smooth function $f : M \to R$ and $f_j = (f(m_i))$, then the vector v^i whose entries are obtained from f by extracting those entries corresponding to points in the neighborhood Γ_i ,then, the matrix vector product $H^i V^i$ gives a d(d+1)/2 vector whose entries approximate the entries of the Hessian matrix, $\frac{\partial f}{\partial U_i \partial U_j}$.

4. Develop quadratic form: Build a symmetric matrix H_{ij} having the entry $H_{i,j} = \sum_l \sum_r ((H^l)_{r,i} (H^r)_{r,j})$, here by H^l we mean the d(d+1)/2 by k matrix associated with estimating the Hessian over neighborhood Γ_i ,where rows r correspond to specific entries in the Hessian matrix and columns i correspond to specific points in the neighborhood.

5. Find approximate null space: Perform an engenanalysis of H, and identify the d+1-dimensional subspace corresponding to the d+1 smallest eigenvalues. There will be an eigenvalue 0 associated with the subspace of constant functions; and next d eigenvalues will corresponding to eigenvectors spanning a d-dimensional space $\hat{V}_d$ in which our embedding coordinates are to be found.

6. Find basis for null spaces: Select a basis for $\hat{V}_d$ which has the property that its restriction to a specific fixed neighborhood Γ_0 provides an orthonormal basis. The given basis has basis vectors $w^1, w^2 ..., w^d$; these are the embedding coordinates.

Being unsupervised, the original HLLE does not make use of class membership of each point. However, a class membership of each sample in the training set is usually known in advance. In this paper, we investigate its extension called Supervised Hessian Locally Linear Embedding

(SHLLE), using class labels of data points in their mapping into a low-dimensional space.

SHLLE modifies step 1 of the original HLLE, while leaving other two steps unchanged. When finding the k nearest neighbors in step 1,SHLLE expands the interpoint distance if the points belong to different classes; otherwise, the distance remains unchanged:

$$\Delta^{'} = \Delta + \alpha \max(\Delta)\Lambda_{ij} \qquad (1)$$

Where $\max(\Delta)$ is the maximum entry of Δ and $\Lambda_{ij} = 0$ if x_i and x_j belong to the same class, and 1 otherwise. Varying α between 0 and 1 gives a partially supervised LLE. When $0 < \alpha < 1$, a mapping could preserve some of the manifold structure but introduces separation between classes. This allows supervised data analysis, but may also lead to better generalization than fully supervised HLLE on previously unseen samples.

C. Implementation and application

In order to get raw information for failure feature extraction, we utilize fault injection to identify which performance metrics are affected by a failure, and then collect empirical data from our test-bed.

As shown in Figure 1, we monitored node-level resource usage as well as application state variables on every node in our distributed test-bed by retrieving the resource-usage and application statistics every second. By periodically examining these variables, we obtained information about the combined resource usage of all of the running processes on the node.

To monitor the network traffic in our system, we have designed and implemented a network sensor based on the NetFlow technique [14]. The information saved in a NetFlow record includes the IP address and port numbers of the source and destination, the protocol type of the traffic, the volume of traffic sent and various other attributes. Our network traffic sniffer allows system analyst to look at higher level trends of traffic flow across the network, rather than becoming overwhelmed by trying to examine each packet that traverses the network. In addition, we gather the network interface and link information by using SNMP protocol [15].The network status variables accessible through SNMP are organized in hierarchies. These data are described by Management Information Base (MIB) [16]. As shown in Figure 2, we collect the raw variables form four groups which are closely related to system performance: interface group, ip group, tcp group and udp group. To reduce the complexity of Implementation, we also adopted a free Java library JPCAP [17] and part of an open source SNMP implementation SNMP4J [18].

It is noticeable that our network information capture is entirely passive on the network, and does not add any overhead to the existing network traffic. We also note that both the node-level and network capture mechanisms are transparent to the application.

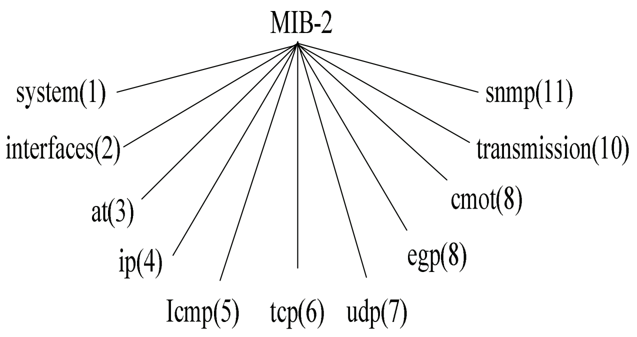

Figure 2. The partial data groups in MIB-2

IV. EVALUATION

In this section, we first introduce the test-bed that is injected various faults and the data collected in our experiments. Then we evaluate the fault injection effect on the system performance and the result of the manifold learning based failure prediction.

A. Test-bed description

Our empirical experiment was conducted in the distributed environment with a file transfer application. In our experiments, we used up to 100 physical nodes (1660MHz processor, 256 Kb cache, 1024MB RAM). The nodes were interconnected by a 100Mbps LAN, as shown in Figure 3. We use a simple two-tier distributed client-server file transfer application as our test application. Once the client sends a file transfer request to the server, in response the server will return more than 2GB of data to the client. In addition, we inject different performance-degrading faults including memory leaks, CPU exhaustion and network interruption, which lead to memory failure, CPU failure and network failure in our system.

In our experiments we found that after the injection of different faults, the transfer rate of our test application per second varies heavily. Due to the lack of fault tolerance mechanism, once injected faults, system will fail in no less than 110 seconds. Therefore there is enough response time after the fault injection to predict the failure and apply lightweight proactive recovery operation.

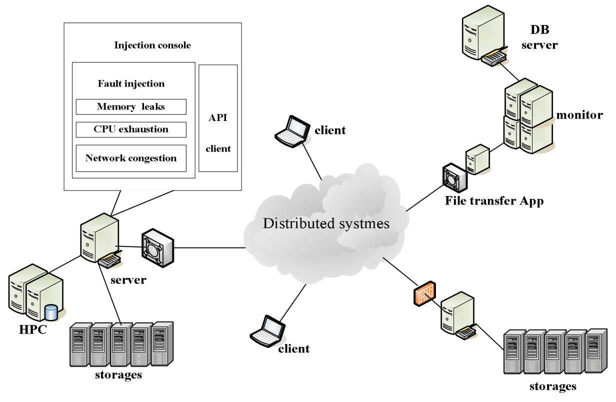

Figure 3. ML-based failure prediction environment

B. Data collection

As discussed in section A, we collect data for our features of interest on every node, and in addition, the network interface and link information. In addition, we collect the data of the same metrics in the fault-free case in order to provide normal samples for failure prediction.

To guarantee enough time for proactive recovery when the prediction is occurred, we collect the data ten second after fault injection. Each experiment covers 12 round-trip client requests and runs for about 10 minutes. Hence, we obtain up to 960 samples for four classes including CPU failure, network failure, memory failure and normal. We randomly select 120 samples of each class as the training set, and the remaining 120 samples as the test set, without overlapping between the two sets. In the following experiments, if not explicitly stated, three failure class data sets are call "cpu_f", "mem_f" and "net_f" respectively.

C. Experimental results

In this section, we present results based on applying the manifold learning based prediction method on our test-bed. In the following experiments, SHLLE is compared with PCA in feature extraction process and their performances on failure prediction are compared. In addition, some other well established classification methods including K nearest neighbor predictor and SVM are also compared.

The classification results of SHLLE and HLLE with different K values are shown in Table 1. In our experiments, we use the precision, recall and F-measure proposed in [11] as the evaluation metrics. These metrics are frequently used for prediction assessment and have been used in similar studies. Specifically, we use recall to represent the true positive rate. To find out whether manifold learning algorithms are sensitive to the value of K, it is tested on the "net_f" data set while K increases from 2 to 20. As shown in the Table 1, the mean precision and recall of SHLLE is little higher than theses metrics of HLLE with a low value of K. And the performance of SHLLE in classification is much better than HLLE when K is larger than 10. Both the performance with different parameter settings and the average performance indicate that SHLLE is more powerful than HLLE to classify the different classes for failure prediction in distributed systems.

We first run each algorithm in order to map the training set into low dimensionality space with different parameter values. And then the samples from the test set are then mapped into low dimensionality space by using the nonparametric generalization solution. The parameters for most of the methods are determined empirically. That is, for each parameter, several values are tested and the best one is selected. When applying KNN algorithm in classification, several values of K from 2 to 40 are tested. Specifically, for SHLLE, different values of α between 0.20 and 0.9 are tested. For SVM, a Gaussian RBF kernel is tested and the best result is reported.

TABLE I. THE COMPARISON OF FAILURE PREDICTON PERFORMANCE OF SHLLE AND HLLE

K	SHLLE			HLLE		
	pre	**rec**	**fpr**	**pre**	**rec**	**fpr**
2	0.224	0.245	0.208	0.247	0.201	0.233
4	0.234	0.319	0.197	0.320	0.210	0.186
6	0.326	0.352	0.190	0.317	0.314	0.218
8	0.438	0.446	0.117	0.323	0.327	0.156
12	0.603	0.543	0.125	0.412	0.326	0.257
16	0.612	0.591	0.112	0.442	0.445	0.216
20	0.723	0.701	0.116	0.408	0.436	0.150
Avg	0.452	0.456	0.152	0.307	0.322	0.202

The performance on failure prediction of SHLLE, PCA, KNN, and SVM on three data sets is shown in Figure 4. According to the average performance, the four methods can be sorted as: SHLLE > PCA > SVM > KNN. We note that SHLLE is the only method of which both the mean precision rate and recall exceed 50%. In contrast, other methods such as PCA and SVM suffer form a low precision and recall in failure prediction. The performance of KNN classifier indicates that without dimensionality reduction process, it is difficult to realize the practical failure prediction in a high-dimension space.

We also note that even the best results of our prediction methods are far from perfect. Our method did encounter false positives and the performance of our prediction still needs to be improved in the future. Nonetheless, by using our prediction method combined with lightweight proactive recovery operations, the failure prediction in large scale systems is of profound practical significance.

V. CONCLUSION AND FUTURE WORK

Online failure prediction has long been considered as a difficult task, especially in large-scale distributed systems. To tackle this problem, in this paper we proposed a novel manifold learning approach to improve the framework of autonomic failure prediction. We have investigated how to make proper online prediction by using a Supervised Hessian Locally Linear Embedding algorithm when the failure-affected metrics are excessive. Finally, experimental results on data from our test bed show that our method can predict more than 60% of the CPU and memory failures, and around 70% of the network failure.

Failure prediction based on the manifold learning is only the first step in autonomic fault handling; it goes without saying that prediction can replace the root cause analysis or

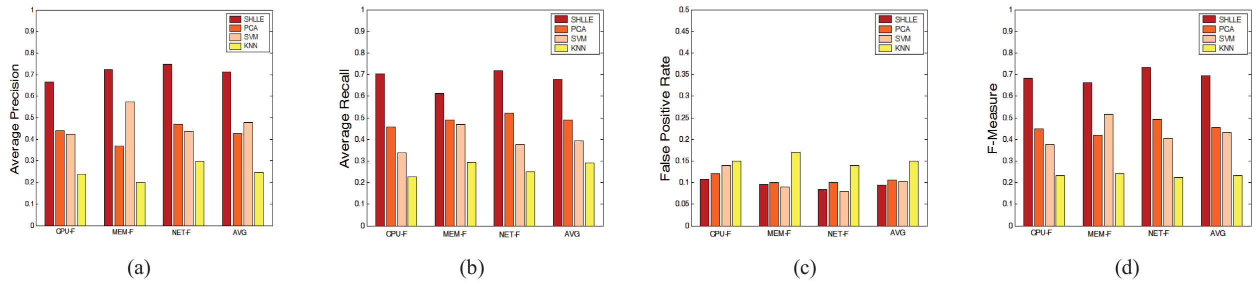

Figure 4. Performance on failure prediction of SHLLE, PCA, SVM and KNN. (a) the average precision, (b) the average recall, (c) the average false positive rate, (d) the average F-measure.

diagnosis. Several possible extensions of the approach include using generalization functions to approximate the nonlinear mappings, exploring more complicated correlation among failures, and developing monitoring tools that can be easily transplanted on different platforms.

ACKNOWLEDGEMENT

This research activity is funded by the National High Technology Research and Development Program of China (grant no. 2007AA01Z401) and the National Natural Science Foundation Major Project of China (grant no. 90718003, 60973027).

REFERENCES

[1] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramanium, R.Sahoo. BlueGene/L Failure Analysis and Prediction Models, In Proceedings of International Conference on Dependable Systems and Networks (DSN06), Philadelphia USA 2006.

[2] Song Fu and Cheng-zhong Xu. Exploring Event Correlation for Failure Prediction in Coalitions of Clusters. In Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (sc07), Reno USA, 2007.

[3] L.D.Solano-quinde, B. M. Bode, Module Prototype for Online Failure Prediction for the IBM BlueGene/L. In Proceedings of IEEE International Conference on Elector/Information Technology (EIT2008), Ames, USA, 2008.

[4] Yanyong Zhang, A. Sivasubramaniam. Failure Prediction in IBM BlueGene/L Event Logs, In Proceedings of the Seventh IEEE International Conference on Data Mining (ICDM2007), Omaha, USA, 2007.

[5] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. Sahoo, J. Moreira, M.Gupta, "Filtering Failure Logs for a BlueGene/L Prototype," In Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN05), Yokohama, Japan, 2005.

[6] A. Csenki. Bayes Predictive Analysis of a Fundamental Software Reliability Model. IEEE Transactions on Reliability, 39(2):177–183, Jun. 1990.

[7] J. Pfefferman, B. Cernuschi-Frias. A Nonparametric Nonstationary Procedure for Failure Prediction. IEEE Transactions on Reliability, 51(4):434–442, Dec. 2002.

[8] A.W.Williams, S. M. Pertet and P. Narasimhan. Tiresias: Black-Box Failure Prediction in Distributed Systems. In Proceedings of IEEE International Parallel and Distributed Processing Symposium.(IPDPS07) Long Beach, USA, 2007.

[9] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S.Trivedi. A methodology for detection and estimation of software aging. In Proceedings of the 9th International Symposium on Software Reliability Engineering (ISSRE 1998), Paderborn, Germany, 1998.

[10] G. A. Hoffmann, M. Malek. Call Availability Prediction in a Telecommunication System: A Data Driven Empirical Approach. In Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems (SRDS 2006), Leeds, United Kingdom, Oct. 2006.

[11] F. Salfner and M. Malek. Using Hidden Semi-Markov models for Effective Online Failure Prediction. In Proceedings of IEEE International Symposium on Reliable Distributed Systems (SRDS07). Beijing, China 2007.

[12] R. Vilalta, C. V. Apte, J. L. Hellerstein, S. Ma, and S. M. Weiss. Predictive Algorithms in the Management of Computer Systems. IBM Systems Journal, 41(3):461–474, 2002

[13] D.L.Donoho, C.Grimes, Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. In the proceeding of the National Academy of Sciences, 2003. 100(10):5591-5596.

[14] R. Sommer , A. Feldman. Netflow: Information Loss or Win. In Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement, Marseille, France, 2002.

[15] RFC1157, Simple Network Management Protocol (SNMP), 1990.

[16] RFC1213, Management information base for network management of TCP/IP-based internets: MIB-II, 1991.

[17] JPCAP:A Java Library for Capturing and Sending Network Packets Http://jpcap.sourceforge.net/ 2009

[18] The SNMP API for Java. Http://www.snmp4j.org/, 2007.