A Holistic Approach to Autonomic Self-Healing Distributed Computing System

Abhishek Bhavsar
Department of Computer Engineering,
College of Engineering, Pune

Ameya More
Department of Computer Engineering,
College of Engineering, Pune

Chinmay Kulkarni
Department of Computer Engineering,
College of Engineering, Pune

Dheeraj Oswal
Department of Computer Engineering,
College of Engineering, Pune

Jagannath Aghav
Department of Computer Engineering,
College of Engineering, Pune

ABSTRACT

Distributed Computing systems are prone to errors and faults and a major amount of time is wasted in maintaining the system and bringing it back to a stable state after a fault. Human resources in the distributed systems architecture currently handle this maintenance. Despite the emergence of ultra-reliable components, failure in distributed computing systems is still an unmitigated problem. As a result of this a lot of resources in the form of money and manpower and efforts in the form of man months are wasted. The proposed mechanism focuses efforts to make a distributed systems environment reliable and robust by proposing an autonomic, self-healing architecture. A holistic approach to the problem is adopted and an architecture that is general enough to be adopted by a wide range of existing systems is proposed. Some of the major challenges include selecting the appropriate actions for healing and reducing the overhead thus making healing lightweight and transparent, yet effective. The proposed system architecture makes use of data mining techniques to generate rules based on gathered system data from logs. The rules are used to make decisions of corrective action and hence carry out the self-healing mechanism.

General Terms

Fault Tolerance, Failure Prediction, Distributed Systems.

Keywords

Autonomic Computing, Self-Healing Systems, Healing Engine, Reliable Systems, Dependable Systems

1. INTRODUCTION

The advent of computers kicked off a race for development of a unified computing platform that could provide for a centralized computing facility, large and reliable storage and increased accessibility. This race brought about revolutionary technologies like Grid Computing, Clustered Systems, Distributed System and many others. Distributed Computing is the most recent development in this field and shows a lot of promise.

There is an ever-increasing demand for large-scale distributed computing systems with tens to hundreds of

thousands of computing nodes that are being designed and deployed. The large scale of distributed computing environments, combined with the ever-growing system complexity, has made reliability a tremendous challenge. Component reliability becomes more difficult with the increasing complexity of the distributed system components as well as growing system size. [1] In order to improve component reliability, considerable research has been done to make distributed computing architectures resilient to faults and to make their applications more robust. The main aim is to create a self-healing cloud architecture that can efficiently detect failures in the system and take the corrective action based on meta-learning techniques.

2. MAIN CONTRIBUTION

In this study, a dynamic meta-learning architecture that can detect possible failures in the distributed computing system has been proposed. These failures are detected with reasonable prediction accuracy on the basis of failure logs over a large period of time. The architecture consists of two basic parts to predict and take corrective action:

- Part one preprocesses and analyzes system event logs. The preprocessed (scrubbed) data is analyzed by means of association rule based machine learning to examine interesting events that may possibly lead to faults. The machine learning techniques are also used to identify failure patterns amongst the different distributed system components. These failure patterns are later used to generate the necessary association rules required in the later stages for accurate failure prediction.
- Part two uses the rules generated by the first part in order to carry out the corrective action and hence prevent failure. This may be done by means of migrating virtual machines running on computing nodes that are predicted to fail to another computing node.

3. HEALING

3.1 Healing – An Overview

Healing is the process of restoring a damaged or diseased system to its original working state, which is free from these problems. Being able to automatically detect and discover faults is of great importance for any healing system. The necessary actions must be taken in order to return to a working and fully functional working state. Distributed computing systems must also incorporate healing in order to ensure efficient working which can quickly, efficiently and accurately recover from a problematic state to its previous working state. The distributed computing environment can also suffer from a varied range of problems and failures. Some of these problems are:

- Security issues at both the client and at the server ends.
- Privacy of the users that have registered themselves with a remote distributed computing system.
- Integrity of the user data that is stored on the servers.
- Theft of the user data stored on servers.
- Loss of the user data stored on servers. Applications to be stored on the distributed system that are infected or are remotely stored on the distributed system with malicious intent.
- The services and resources provided by a distributed computing infrastructure are not limited by geographic boundaries. The distributed computing systems provide these services to their clients through a single interface thus making it very difficult to locate the data of different users on the physical storage at the distributed server end.

All the above-mentioned problems make it absolutely necessary to develop an efficient healing system for the distributed computing system.

3.2 Faults

Any system that is working perfectly can be susceptible to a large number of faults. Depending on the tasks executed by that system, these faults might take a varied number of forms. The combined effect of these faults is a decrease in the productivity of the system and hence a drop in the system efficiency. In layman terms, the given system no longer functions as it used to before the occurrence of the faults in the system. These faults may occur at different levels in the system architecture. Furthermore, certain faults can trigger the occurrence of subsequent faults and this can be disastrous. A fault in a system is thus a malfunction that leads to a certain deviation from the expected behavior of the system. If the case of a system of inter-connected computers were to be considered, faults may occur due to a large number of factors, including hardware failure, software bugs, software failure and network problems. Three types of faults are observed in a typical distributed system of computers:

- Transient faults: These faults occur once and then disappear. For example, a network message doesn't reach its destination but does when the message is retransmitted after some period of time.
- Intermittent faults: Intermittent faults fault that are reoccurring. These are the most irritating of faults and occur mainly due to component failures or improper inter-component operation, like a loose connection.
- Permanent faults: This type of failure is persistent and it will continue to exist as long as the faulty system component is repaired or even fully replaced in extreme cases. Examples of this fault are disk head crashes, software bugs, and burnt-out power supplies.

It is thus essential for any system to incorporate techniques to resolve these faults as quickly and effectively as possible. In general, a fault tolerant system is what is required.

3.3 Fault Tolerance

The basic approach to building fault tolerant systems is redundancy. Redundancy may be applied at several levels.

- Information redundancy: Information redundancy is used to provide fault tolerance by replicating or coding the data. For example, a Hamming code is used to provide extra bits in the data in order to recover a certain ratio of failed bits. Other important samples used to provide information redundancy are parity memory, ECC (Error Correcting Codes) memory and ECC codes on data blocks.
- Time redundancy: Time redundancy achieves fault tolerance by performing an operation several times. Retransmissions in a reliable point-to-point and the use of time-outs along with group communication are examples of time redundancy. This form of redundancy is extensively useful in the presence of transient or intermittent faults. It is of no use with permanent faults. An example is the retransmission of TCP/TP packets.
- Physical redundancy: Physical redundancy deals with devices rather than data. Extra equipment is added to enable the system to tolerate the loss of some failed components. RAID disks and backup name servers are examples of physical redundancy.

There are many challenges with regard to the implementation of fault tolerant architecture for any distributed system or autonomic computing system. Some of these challenges are:

- Implementation of autonomic fault tolerance techniques is required for multiple instances of any application that is running on the several virtual machines that are provided via the distributed computing infrastructure.
- Fault tolerant techniques must be developed which are integrated with the existing workflow scheduling algorithms that have been implemented for the underlying autonomic system.
- It is important that a great level of reliability and availability of multiple distributed computing providers with independent software stacks be ensured.
- Autonomic fault tolerant methods must react in accurate synchronization among the various interacting distributed systems; otherwise the solution itself can lead to future faults in the system. It is quite obvious that the techniques that can be developed for automatic fault tolerance are accompanied by a large number of limitations. In the foresight of various users (clients) registered with a distributed system, requesting services, it is thus a great ordeal to ensure efficient provision of services without error. Fault tolerant mechanisms cannot be relied upon regardless of them being automatic.

It is extremely difficult to synchronize fault repairs amongst the various interconnected nodes of our autonomic distributed computing system, thus manual intervention in the healing process must be kept to a minimal level. Apart from synchronization, many other aspects must also be considered, including automatic and accurate load balancing in the unfortunate case of the failure of a node or any component of the distributed system. It is this that has led to the growing demand for an autonomic distributed computing system that is capable of self- healing keeping in mind that the system is not completely fault tolerant and impervious to faults.

3.4 Need for Prediction

The Distributed Computing environment may be distributed over geographical locations. Also, inherent in the definition of distributed computing systems, is the notion of leasing resources to third parties. The distributed systems service provider needs to ensure uninterrupted and failure proof service. Thus the distributed nature and service liabilities entail a robust system that is reliable and requires minimum human intervention. One of the most fundamental and essential features that must be incorporated to achieve these goals is failure prediction. Failure prediction is an ensemble of various analytical techniques that work together closely to predict failures and thus trigger healing action.

3.5 Self-Healing

Ever since development of modern computers it has become very difficult to rectify the system faults and manage recovery from malicious attacks due to the increase in complexity of the systems. All these factors resulted in the study in the field of autonomic computing and have explored the concept of self-healing systems. Autonomic computing is a self-managing computing model named after, and patterned on, the human body's autonomic nervous system. Self-healing in autonomous computing is described as the process to free people from discovering, recovering, and failures. Self-healing systems are expected to heal themselves at runtime in response to any change in environment or operational circumstances. Thus, the goal of self-healing is to prevent disastrous failure through prompt execution of certain proposed actions. A self-healing mechanism that is expected to monitor, diagnose, recover from faults and regain normative performance levels independently is needed. The Self-healing technology enhances the system reliability by removing the need for human operation, as human configuration and maintenance of complicated systems makes the system more vulnerable to errors. Conventional ways to eliminate these errors would include log-based level, model-based level, and component-based level approaches. These approaches do support some parts of the self-healing process but not the whole process that includes monitoring, filtering, translation, analysis, diagnosis, decision and healing.

4. PROPOSED PREDICTION SCHEME

The failure prediction scheme incorporates various data mining techniques to predict failures by generating rules. This module is independent of the autonomic healing engine and thus can be upgraded with new and better algorithms transparently.

A high level diagram of the proposed fault prediction scheme is proposed below. The scheme involves two parts: data scrubbing and the other for fault prediction. The RAS logs from the Blue Gene/L System available at ANL were used.[2][3][4][5]

4.1 Data Scrubber

The RAS logs cannot be used as is for data mining because of the unevenness in reporting the same type of error messages. The Data Scrubber handles all the actions of cleaning data so as to make it usable for mining. This step also helps in reducing the amount of data storage space required to store the statistical data. Upon completion, the data scrubber intends to provide a list of unique events for failure prediction

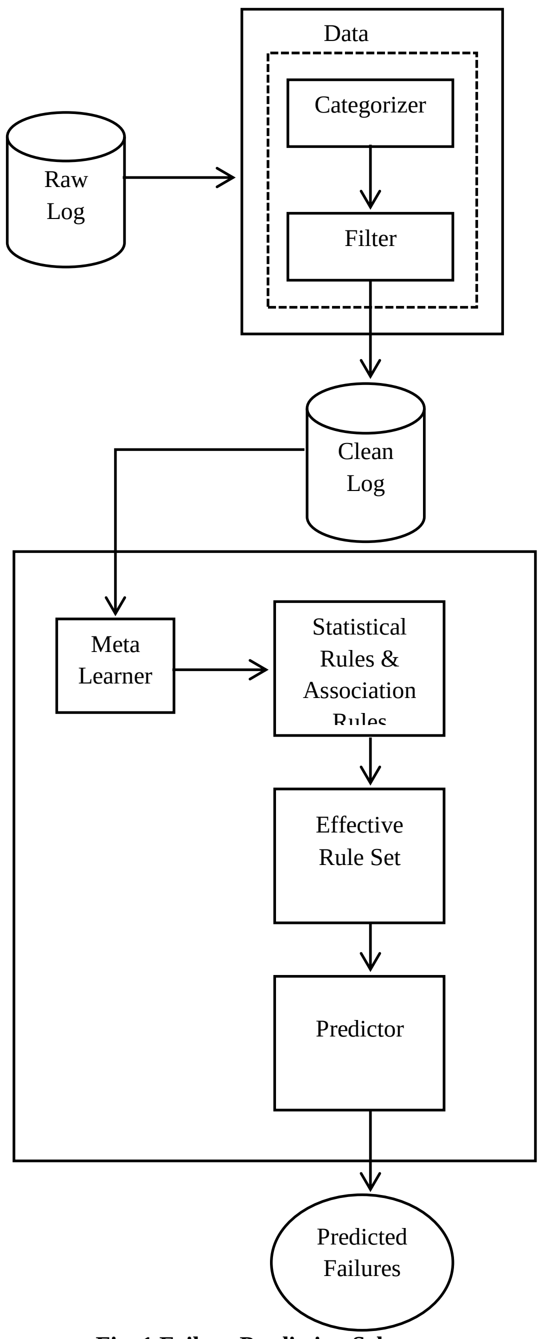

Fig. 1 Failure Prediction Scheme

4.1.1 Event Categorization

The RAS log, has a lot of information and because of the inherent distributed nature of the Blue Gene/L System, the logs cannot be used as is. The logs originate from a variety of different facilities and carry a lot of information that needs to be made meaningful to the data-mining program. This can be achieved by categorizing events. Categorization of events maps the problem to a binary model where the particular event either occurs or not. Event categorization is also done over the fatality of the event. This helps in recognizing failures efficiently.

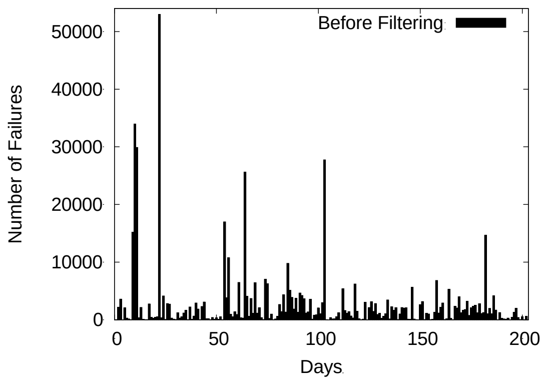

Fig. 2 Failure Count Before Filtering

4.1.2 *Event Filtering*

The distributed nature of the system causes a lot of duplication in the logs.[6][7] As the job is distributed across nodes, the same job reports the same type of events multiple times. By studying event duplication times, a 300 sec threshold for temporal compression of data was decided. Event logs that appeared from the same location within a 300 sec window having the same category field were reported only once.

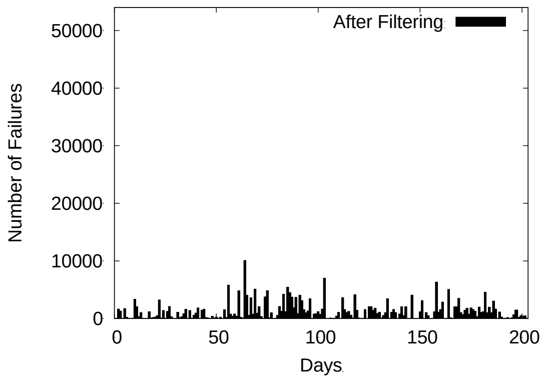

Fig. 3 Failure Count After Filtering

4.1.3 *Prediction*

The scrubbed data can now be processed by the Prediction Scheme mechanism in the Fault Prediction Engine, to analyze the data logs for failure patterns and irregular entries. The techniques of Association Rule Based Learning are used for this prediction. The scrubbed data logs are fed to the Weka Data Mining Tool. [8] Weka is configured to find associations amongst the entries in the data log. The Apriori algorithm is preferred to other Association Rule Based Learning algorithms because it can be easily configured to work with the large datasets that are used in this research work. [9][10] The parameters of "Apriori Associate" are set to a minimum support of 0.01 and a threshold confidence value of 0.1. Once Weka has finished the processing of the data logs, it returns a set of rules. These rules are supplied with confidence values and those with high confidence values for failure events are predicted to be possible future failures. Those events that can lead to failures with a high confidence value trigger the Automatic Healing Engine to migrate VMs and their processes from this compute node that is likely to fail to a compute node that is processing normally.

5. PREDICTION ALGORITHM

Based on the effective rule-set create two lists: Triggered Failures List and Triggering Event List

$$TE-List = \{f_i \rightarrow \{e_{i1}, e_{i2}, \dots, e_{ik}\} : \ 1 \le i \le N_f \}$$

$$TF-List = \{e_m \rightarrow \{f_{m1}, f_{m2}, \dots, f_{mn}\} : \ 1 \le m \le N_e \}$$

Where f_i is a fatal event and e_j is an event (non fatal or fatal)

During prediction, when an event e occurs:

1. Append e into the prediction event set

$$E = \{e_1, e_2, \dots, e_n\}$$

where the events are sorted in an increasing order of their occurrence items, and remove e_i when

$$|T_e - T_i| > T_{predict\_window}$$

2. Obtain potential failures that may be triggered by e according to the $TF-List : e \rightarrow \{f_1, f_2, \dots, f_k\}$
3. For each failure in the set of $\{f_1, f_2, \dots, f_k\}$, go through its event list according to the $TE-List : f_i \rightarrow \{e_{i1}, e_{i2}, \dots, e_{ik}\}$
4. If $\{e_{i1}, e_{i2}, \dots, e_{ik}\} \subseteq E$, then produce a warning that the failure f_i may occur within $T_{predict\_window}$

6. EVALUATION METRICS

6.1 Precision

Precision is the probability that a (randomly selected) retrieved document is relevant.

$$P = \frac{T_p}{T_p + F_p}$$

6.2 Recall

Recall is the probability that a (randomly selected) relevant document is retrieved in a search.

$$R = \frac{T_p}{T_p + F_n}$$

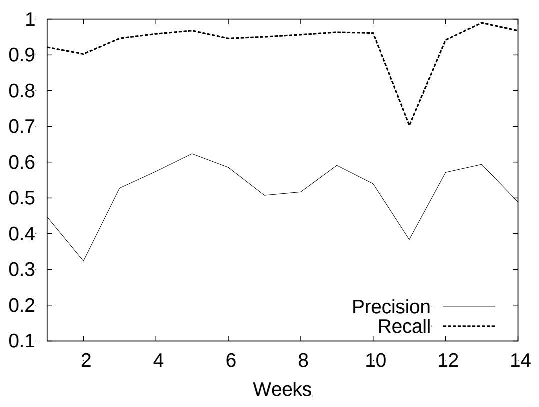

Fig. 4 Precision And Recall at 0.4 Confidence

7. CONCLUSION

In this paper, a self-healing prediction engine for large-scale Distributed Systems is proposed. The case studies with the Blue Gene/L systems logs have shown good accuracy by correctively predicting high precision and recall values. The study has also shown that the approach is well suited for predicting the failures and taking the corrective healing action before the occurrence of the failure. The aim in future would be to come up with a more specific healing mechanism. First being the prediction window. The size of the prediction window in this approach is fixed. The next aim would be to make this window size dynamic. This would lessen the training cost without hampering the prediction accuracy. In this approach the Apriori Algorithm was used as the data mining method. Hence, it is planned to use various other data mining techniques, such as, decision tree, Filtered Associator, FPGrowth, Predictive Apriori, etc. thereby making the prediction scheme more efficient.

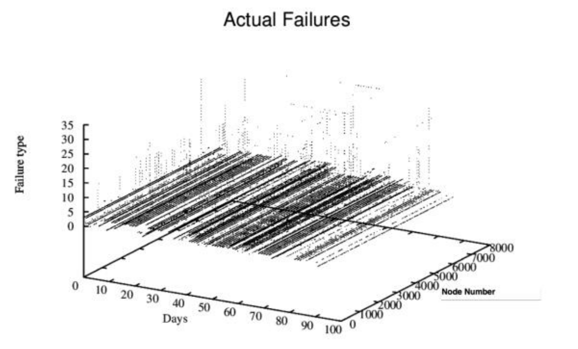

Fig. 5 Plot of Actual Failure Data

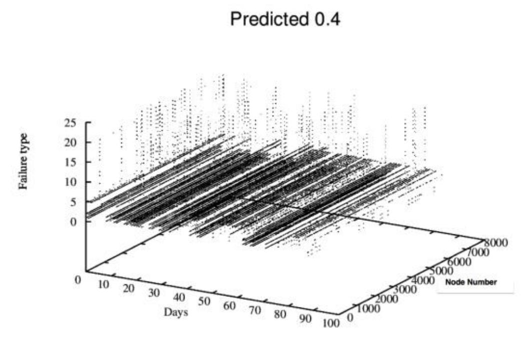

Fig. 6 Plot of Failures Predicted at 0.4 Confidence

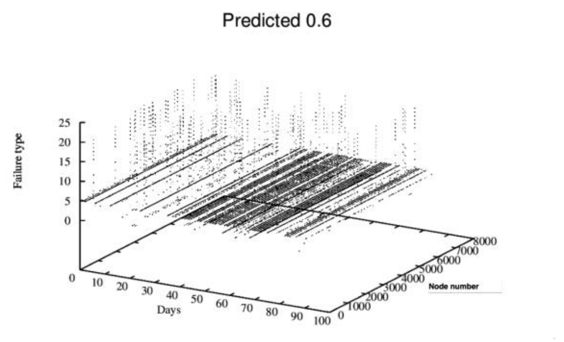

Fig. 7 Plot of Failures Predicted at 0.6 Confidence

8. ACKNOWLEDGMENTS

Our thanks to the staff at the data center and the Department of Computer Engineering at College of Engineering Pune for supporting us to carry out the research work.

9. REFERENCES

[1] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam, “Critical event prediction for proactive management in large-scale computer clusters,” in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD ’03. New York, NY, USA: ACM, 2003, pp. 426–435. [Online]. Available: http://doi.acm.org/10.1145/956750.956799

[2] Y. Liang and Y. Zhang, “Failure prediction in ibm bluegene/l event logs.”

[3] T. B. Team, T. Domany, M. Dombrowa, W. Donath, M. Eleftheriou, C. Erway, J. Esch, J. Gagliano, A. Gara, R. Garg, R. Germain, M. Giampapa, B. Gopalsamy, J. Gunnels, B. Rubin, A. Ruehli, S. Rus, R. Sahoo, A. Sanomiya, E. Schenfeld, M. Sharma, S. Singh, P. Song, V. Srinivasan, B. Steinmacher-burow, K. Strauss, C. Surovic, T. Ward, J. Marcella, A. Muff, A. Okomo, M. Rouse, A. Schram, M. Tubbs, G. Ulsh, C. Wait, J. Wittrup, M. B. (ibm Server Group, K. D. (ibm Microelectronics, and L. Kissel, “An overview of the bluegene/l supercomputer,” 2002.

[4] A. Gara, M. Blumrich, D. Chen, G.-T. Chiu, P. Coteus, M. Giampapa, R. Haring, P. Heidelberger, D. Hoenicke, G. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas, “Overview of the blue gene/l system architecture,” *IBM Journal of Research and Development*, vol. 49, no. 2.3, pp. 195–212, 2005.

[5] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. Sahoo, “Bluegene/l failure analysis and prediction models,” in *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, 2006, pp. 425–434.

[6] J. Hansen and D. Siewiorek, “Models for time coalescence in event logs,” in Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on, 1992, pp. 221–227.

[7] S. Fu and C.-Z. Xu, “Exploring event correlation for failure prediction in coalitions of clusters,” in *Supercomputing, 2007. SC ’07. Proceedings of the 2007 ACM/IEEE Conference on*, 2007, pp. 1–12.

[8] "Data mining and machine learning using weka." http://www.cs.waikato.ac.nz/ml/weka/.

[9] S. B. Aher and L. L.M.R.J, "Article: A comparative study of association rule algorithms for course recommender system in e-learning," *Inter- national Journal of Computer Applications*, vol. 39, no. 1, pp. 48–52, February 2012, published by Foundation of Computer Science, New York, USA.

[10] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. of 20th Intl. Conf. on VLDB,* 1994, pp. 487–499.