Online Failure Forecast for Fault-Tolerant Data Stream Processing

Xiaohui Gu[§], Spiros Papadimitriou[‡], Philip S. Yu[*], Shu-Ping Chang[‡]

[§]*North Carolina State University, Raleigh, NC*
[§]gu@csc.ncsu.edu
[‡]*IBM T.J. Watson Research Center, Hawthorne, NY*
[‡] {spapadim, spchang}@us.ibm.com
[*]*University of Illinois at Chicago, Chicago, IL*
[*]psyu@cs.uic.edu

***Abstract*— In this paper, we present a new online failure forecast system to achieve predictive failure management for fault-tolerant data stream processing. Different from previous reactive or proactive approaches, predictive failure management employs failure forecast to perform informed and just-in-time preventive actions on abnormal components only. We employ stream-based online learning methods to continuously classify runtime operator state into normal, alert, or failure, based on collected feature streams. We have implemented the online failure forecast system as part of the IBM System S stream processing system. Our experiments show that the on-line failure forecast system can achieve good prediction accuracy for a range of stream processing software failures, while imposing low overhead to the stream system.**

I. Introduction

Data stream management systems can be useful for many emerging applications such as sensor data analysis and network traffic monitoring. Data stream processing requires continuous system operation that makes automatic failure management imperative for any stream management system. On the other hand, stream processing applications are often long-lived, which provides opportunities for the system to observe processing patterns and perform meaningful failure predictions.

Previous failure management work (e.g., [6], [2], [4]) can be classified into two categories: (1) *reactive* approach that takes recovery actions after a failure happens, and (2) *proactive* approach that takes advance preventive actions such as backup for *all* system components at *all* time. The reactive approach does not have any preventive cost but can incur significant failure penalty (e.g., losing important query results) for stream applications. The proactive approach offers better fault tolerance but can be too costly for stream systems that are often under resource constraint pressure. To address the problem, we explore a new *predictive* failure management approach that employs *online failure forecast* to perform just-in-time, preventive actions (e.g., backup) on abnormal components only instead of all components.

To predict failures, the system continuously monitors the behavior of each component (e.g.., CQ operator) as captured by a *feature stream* that consists of periodically collected system log data (e.g., available memory, free CPU time, virtual memory page-in/page-out rates, etc.) and dynamically generated software instrumentation data (e.g., tuple processing time, buffer queue length, etc.). The failure forecast system consists of a set of failure prediction models called predictors that continuously classify received feature stream tuples into three states: *normal*, *alert* and *failure*. One way to describe the failure state is via a failure predicate (e.g., "processing time $>$ 50ms") that characterizes *what* we are trying to predict. The alert state corresponds to a *warning* region where the predictor will raise a failure alert, which may be used to trigger the appropriate failure prevention actions. The warning region is defined by a dynamically selected *pre-failure interval* which "precedes" the failure.

We need to address a set of new challenges to achieve efficient failure forecast for data stream processing systems. First, failure forecast should be *light-weight* since normal continuous query processing can be resource-intensive by itself. Second, data stream processing demands an *online* failure forecast scheme, which can continuously learn the runtime behavior of different operators to raise advance alert before a failure actually happens. Third, dynamic stream environments require *adaptive* failure forecasting that can achieve a desirable tradeoff between correct predictions and false-alarms under time-varying stream workloads and system conditions.

We employ stream-based decision tree classification methods to achieve light-weight, online failure forecast. The system raises failure alerts when a set of consecutive feature tuples are classified as alert or failure. Each decision tree is dynamically updated using feedback from the system that provides true labels (i.e., normal, alert, failure) for retained feature tuples. We introduce online failure prediction adaptation methods to cope with dynamic stream environments. Each failure predictor maintains an ensemble of decision trees using a range of pre-failure intervals. Each tree operates at different points in the proactive/reactive spectrum and the predictor dynamically switches to the best decision tree based on a failure forecast reward function.

We have implemented the online failure forecast system as part of the predictive failure management framework and tested it on the IBM System S stream processing infrastructure [5]. We have implemented several query networks that

can process real data streams using different continuous query (CQ) operators (e.g., join, split, merge). The experimental results show that our online failure prediction schemes can achieve good prediction accuracy for several common software faults.

II. Online Failure Forecast

A. Approach Overview

To perform online failure forecast, the predictor continuously collects feature metrics and classifies received feature tuples into three possible states: *normal*, *alert*, and *failure*. The *failure* state can be described with a predicate that characterizes *what* we are trying to predict and may be related to SLO violations (e.g., "processing time $>$ 50ms," or "number of output tuples $<$ 100/sec."). The *alert* state corresponds to a *pre-failure interval* of duration PF that "precedes" the failure. Everything else belongs to the normal state.

Our online failure forecast system mainly consists of *monitoring* and *analysis* components, illustrated by Figure 1. The monitoring components are typically co-located with query components and are responsible for collecting feature streams by periodically sampling system log files and software instrumentation sensors. The sampling rate is dynamically adjusted based on the feedback (i.e., prediction result) from the analysis components. The analysis components maintain a compact training data set using reservoir sampling and perform failure prediction by classifying received feature tuples into different states. To achieve best failure prediction reward in dynamic stream environments, the system labels training data using different pre-failure intervals PF to create an ensemble of decision tree classifiers. These classifiers "compete" by essentially operating at different trade-off points on the reactive/proactive spectrum. At any given moment, the tree with the largest reward value is selected as the primary classifier. This scheme allows us to inexpensively switch to the best classifier appropriate for the current stream conditions. The analysis components can be located on a different host from the monitored component for fault-tolerance and load balancing. The failure forecast system generates alerts that allow other predictive failure management components to take preventive actions on abnormal components and, conversely, receives feedback from them to calculate prediction accuracy (i.e., A_D and A_F) for performing adaptations.

B. Classification Method

Among many statistical machine learning methods such as decision trees, Gaussian mixture models, or support vector machines, we chose decision trees since they produce rules with direct, intuitive interpretation by non-experts. Thus, the predictor can not only raise advance failure alerts but also provide cues for possible failure causes. Each decision tree classifier is trained on historical measurement data, which are appropriately labelled with "normal", "alert", or "failure". The system can label feature tuples as normal or failure based on the failure predicate. Then, a set of points preceding the failure incidents within the pre-failure interval are labelled as "alert"

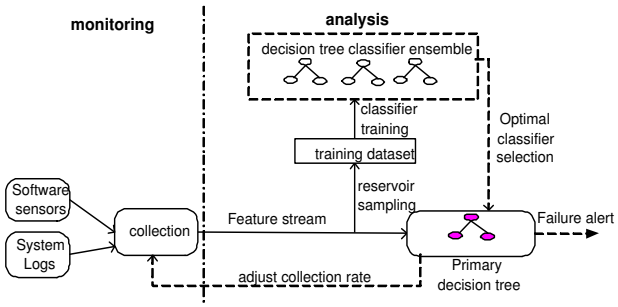


Fig. 1. Online failure forecast using stream-based learning methods.

The decision tree is trained using feature tuples from all three states. Periodically, as the history grows based on feedback from the system, the decision tree is updated if its accuracy is low. For state classification, decision trees essentially apply a sequence of threshold tests on the features. The inductive bias of decision trees consists of a restriction to isothetic (i.e., axis-parallel) decision boundaries, as well as a preference for the shortest trees. The first assumption is reasonable in our context, since system state can be successfully characterized by such isothetic boundaries (i.e., threshold tests). Additionally, seeking the smallest set of such tests that can separate the states is desirable for reasons of simplicity, interpretability as well as performance.

C. Feature Selection

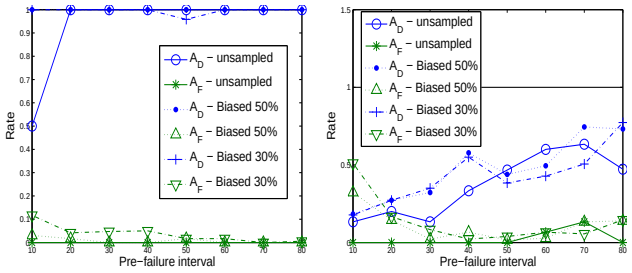
Examining a single metric in isolation is not desirable because the metric(s) involved in the failure predicate may be insufficient to capture "abnormal behavior" early enough to issue an advance alert. For example, say we want to issue an alert for query processing hot-spots, which might be characterized by, e.g., "processing time $> 50\text{ms}$", and which may be due to memory or buffer exhaustion. The problem is that this predicate is often violated suddenly. Consequently, it is impossible, even for a human, to predict processing hot-spots early enough by examining the processing time alone. However, there are often other metrics that can provide early indication of potential problems. In this example, available free memory and buffer queue length are two such metrics, which are gradually exhausted. Therefore, an intelligent predictor wishing to predict processing hot-spots, would actually watch out for suspect trends not in processing time, but in memory utilization (and possibly other metrics, e.g., number of transactions processed, etc). In order to effectively and automatically discover the appropriate features for prediction, the classifier has to incorporate multiple features in the early stages. Thus, the initial set of features is a superset of those containing information that can classify what is normal and what isn't. From an initial large set of features, we will select just those appropriate for state classification. An additional benefit of decision trees is that they inherently do this, by seeking the shortest possible tree that explains the data. Additionally, we employ ten-fold cross-validation to select those features with best predictive power.

III. EXPERIMENTS

We have implemented the online failure forecast scheme in the IBM System S distributed stream processing system [3]. The online decision tree classifier is implemented based on the canonical C4.5 decision tree software package. The stream processing system consists of about 250 IBM blade servers connected by Gigabit networks. Each server host is equipped with Intel Xeon 3.2GHZ CPU and 2 to 4 GB of memory. All of our experiments are conducted on the stream processing cluster. For failure prediction, the system continuously collects system-level metrics (e.g., free CPU time, available memory) and application-level metrics (e.g., input/output data rate, input queue length, per-tuple processing time). Our case study query networks are taken from the System S reference application [3]. We have tested our system using a range of software failures caused by common program bugs such as memory leak bug, infinite loop bug, and buffer deletion bug.

We first use the query network to process the network traffic data taken from the Internet Traffic Archive [1]. The software faults are injected on a set of replicated join operators at different time instants and under different workload conditions. Figure 2(a) shows the detection rates (A_D) and false-alarm rates (A_F) achieved by different decision tree classifiers for predicting the failures caused by the memory leak fault. These decision trees are trained using different pre-failure intervals ranging from 10 to 80 seconds. We also compare the performance of the classifier using full training data with that of the classifiers using 50% biased sampling (i.e., retaining half of the training data) and 30% biased sampling (i.e., retaining 30% of the training data). We observe that for the network traffic data, our failure prediction models can achieve almost perfect predictions (i.e., 100% detection rate and 0% false-alarm rate) by employing proper pre-failure intervals. We also observe that bias-sampling can effectively maintain prediction accuracy while greatly reducing the training overhead. It is also interesting to note that the trees trained on the bias-sampled data can sometimes achieve a higher detection rate than the tree trained on the full data, without much change in the false alarm rate. The increase in A_D is because the sampling is biased towards non-normal points, making the classifiers less conservative in raising an alert. However, A_F does not increase correspondingly in this case because the normal behavior is better separated from the faulty behavior in the measurement space. Generally speaking, bias-sampling can maintain the detection rates of different classifiers with a slight increase in false-alarm rates for classifiers trained with small pre-failure intervals.

We then conduct the second set of experiments using the video stream data taken from the NIST TRECVID data set. Figure 2(b) shows the prediction performance of different decision tree classifiers using a range of pre-failure intervals. We observe that our failure prediction models can still achieve reasonably good prediction accuracy by choosing the best classifiers with proper pre-failure intervals. However, the prediction models are generally less perfect for the video stream data than for the network traffic data. The reason is that the differences between normal feature streams and abnormal feature streams become more subtle under lower stream rates.



(a) Network Join Operator (b) Video Join Operator

Fig. 2. Prediction accuracy for faulty join operators with the memory leak bug.

IV. CONCLUSION

In this paper, we have presented a novel failure forecast system to enable predictive failure management for fault-tolerant continuous stream processing. We view our work as the first step toward providing light-weight failure forecast in an online, streaming setting. We employ stream-based decision tree classifiers for simple, fast and effective characterization of failure and alert states. The classifiers can be continuously updated, based on feedback from the failure management framework. We have implemented a prototype of the online failure forecast system as part of the predictive failure management framework for the IBM System S stream processing system. The experimental results show that our failure prediction schemes can achieve good prediction accuracy for failures caused by several typical stream processing program bugs, while imposing low overhead to the stream processing system.

V. ACKNOWLEDGEMENT

The work was done when all authors are with IBM research. We thank Nagui Halim, the principal investigator of the System S project, and Lisa Amini, the prototype technical leader, for providing us with invaluable guidance throughout the development of the system. We also thank Henrique Andrade, Yoonho Park, Philippe L. Selo and Chitra Venkatramani for their help.

REFERENCES

[1] Internet Traffic Archive. *http://ita.ee.lbl.gov/*.
[2] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. Fault-Tolerance in the Borealis Distributed Stream Processing System. *Proc. of SIGMOD*, 2005.
[3] K.-L. W. et al. Challenges and Experience in Prototyping a Multi-Modal Stream Analytic and Monitoring Application on System S. *Proc. of VLDB*, 2007.
[4] J.-H. Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, and S. Zdonik. High Availability Algorithms for Distributed Stream Processing. *Proc. of ICDE*, 2005.
[5] N. Jain and et al. Design, Implementation, and Evaluation of the Linear Road Benchmark on the Stream Processing Core. *Proc. of SIGMOD*, 2006.
[6] M. Shah, J. Hellerstein, and E. Brewer. Highly-available, fault-tolerant, parallel dataflows. *Proc. of SIGMOD*, 2004.