Common Framework For Automated Diagnosis on Distributed Information System

Iping Supriana Suwardi1*, Juwairiah2

1 Bandung Institute of Technology, Jalan Ganesa 10, Bandung 40132, Indonesia
2 UPN"Veteran"Yogyakarta , Jalan Babarsari 2 Tambakbayan Yogyakarta, Indonesia

Many information systems, suffer from a common problem : when system fail to function properly, it is often difficult to determine which part of the system is source of the problem. The time that is needed to detect and determine the sources of the problem can take days, but they are repaired quickly once found. This cause late recovery from failure, so the system have low availability and become unreliable. This paper will learn some approach to diagnose root cause of system failure without detailed knowledge about system structure and propose a common framework to do diagnosis based on these approach. We deploy sensors inside system and the sensors monitor the system behavior and runtime properties, that give information to help diagnose faults. Combining with the information of success or failure at the application layer, and using statistical analysis, user can infer which sensors indicate the most relevant data may contributed to the failure. Thus, it can do fast detection in order to aid system recovery process faster.

1. Introduction

Many information systems, especially web-based applications, are composed of a number of communicating components. These are often structured as distributed systems, with components running on different processors or in different processes1). For example, a multi-tiered system start with requests from web clients that flow through a web-server front-end and then to a web “application server,” which in turn makes calls to a database server, and perhaps additional services (authentication, name service, credit-card authorization, customer relationship management, etc.).

Information system are growing more and more complex, toward beyond the limit that human can handle. Complex systems exhibit performance problems that often grow out of the system complexity. Many information system suffer from the same problem of failure : when the system fail to function properly, often hard to determine which part of system as source of problem. Source of problem in software system is usually called fault. Application level failure can sometimes take days to detect, though they are repaired quickly once found. This can cause late recovery from failure, so the system have low availability and become unreliable.

A common approach is to build an expert system that incorporates a priori model of structures and behaviors, and use rules to guide diagnosis. However, it is very hard to build the models and the rules, and more expensive to maintain them manually when the system is evolving.

Thus, it is necessary to propose another approach to solve this diagnosis problem. Faults are detected and causes are identified without detailed knowledge of the system structures and correct behavior. Models are induced from a statistical learning perspective. Such an approach is promising because it requires less human intervention and therefore is more automatic.

2. Contribution

Main purposes of this paper are:

1. Learn three different machine learning approach to fault diagnosis in different problem area
2. Make a common framework that can be used by different approaches to fault diagnosis based on similarity three different machine learning approach.

3. Fault Diagnosis

Fault diagnosis is a central aspect of fault management for networked system4). Since faults are unavoidable in systems, their quick detection and isolation is essential for the robustness, reliability, and accessibility of a system. In large and complex communication networks, automating fault diagnosis is critical.

These are several basic concepts in the field of fault management5):

• Event, defined as an exceptional condition occurring in the operation of hardware or software of a managed network, is a central concept pertaining to fault diagnosis
• Faults (also referred to as problems or root causes) constitute a class of network events that can cause other events but are not themselves caused by other events. Faults may be classified according to their duration time as: 1) permanent, 2) intermittent, and 3) transient.
• Error is defined as a discrepancy between a computed, observed, or measured value or condition and a true, specified, or theoretically correct value or condition. Error is a consequence of a fault. The term failure is used to denote this type of an error.
• Symptoms are external manifestations of failures

The process of fault diagnosis usually involves three steps:

• Fault detection : a process of capturing on-line indications of network disorder provided by malfunctioning devices in the form of alarms.
• Fault localization (also referred to as fault isolation, alarm/event correlation, and root cause analysis) : a set
of observed fault indications is analyzed to find an explanation of the alarms.

- **Testing**: a process that, given a number of possible hypotheses, determines the actual faults.

4. Existing Fault Monitor

Information system’s failures manifest in many ways, one particularly pernicious class of failures is the application-level failure whose only obvious symptoms are changes in the semantic functionality of the system. A simple system can be modeled as a layered stack of software, where the lowest layers are the hardware and the operating system, and the highest layer is the application, with various other layers in between (e.g., libraries, middleware software, standard protocols). In this model, an application-level failure manifests solely in the application layer, though the cause of the failure might be in another layer.

Detection methods to monitor existing fault fall into three categories:

- **Low-level monitors**: are machine and protocol tests, such as heartbeats, pings and HTTP error code monitors.
- **Application-specific monitors**: such as automatic test suites, can catch highlevel failures in tested functionality.
- **Business-metric monitors**: watch simple statistics about the gross state of high level metrics relevant to the core business of the Internet service. These monitors are essentially watching user behavior, and they can generate false alarms due to external events, such as holidays or disasters.

Monitor has to be sure it is monitoring all critical components of the system. The challenge is that the components and their criticality are application and system-specific. To be most useful in a real system, a monitoring technique should have the following properties:

- **High accuracy and coverage**: A monitoring service should correctly detect and localize a broad range of failures. Ideally, it would catch never-before-seen failures anywhere in the system. In addition, a monitor should be able to report what part of the system is causing the failure.
- **Few false alarms**: The benefit provided by early detection of true failures should be greater than the effort and cost to respond to false alarms.
- **Deployable and maintainable**: A monitoring system must be easy to develop and maintain, even as the monitored application evolves.

Approach for monitoring a system for anomalies (likely failures) can be divided into a straight-forward three-stage process:

1. **Observation**: capture the runtime properties and runtime path of each request served by the system: This path is the ordered set of coarse-grained components, resources, and control-flow used to service a client's request. From these paths, we extract low-level behaviors.

2. **Learning**: build a reference model of the fault-free behavior of system, under the assumption that most of the system is working correctly most of the time.

3. **Detection**: analyze the current behavior of the system and search for anomalies with respect to learned reference model.

5. Machine Learning Approach

Fault diagnosis can be done in many area in information system, e.g. bug isolation, localization fault in internet services, monitoring performance problem, using machine learning approaches: logistic regression, decision tree, and Tree-Augmented Naïve Bayesian Network (TAN) respectively.

5.1 Bug Isolation using Logistic Regression Approach

It is well-known that bugs exist in computer programs. There are two types of bug: deterministic and non-deterministic bug. Deterministic bugs are quite common, they are generally easier to find and fix using any methods than non-deterministic bugs.

Liblit et al. proposed statistical bug isolation to address the problem of non-deterministic bug\(^4\). Deterministic bug could be fixed by elimination strategy, and non-deterministic bug could be fixed by logistic regression, one of machine learning approaches. In their framework, they monitor and sample code-level behavior, and analyze the correlation with program failures. The code-level behavior is abstracted as predicates (i.e. assertions), which include signs of function return values, conditional branches, pointer arithmetic, etc. These predicates serve as the “wild guesses” of potential bug sources. Statistical analysis can discover the correlations between bugs and a small number of predicates, therefore it helps the programmers to eventually pinpoint the bugs.

Using formula “penalized log likelihood” and iterative algorithm “gradient-ascent” to estimate \(\beta\) coefficient. Once the model has been trained, the predicates with the largest \(\beta\) coefficient indicate the most suspicious lines of code to look for bugs.

5.2 Localization Fault in Internet Service Using Decision Tree

Failure often occur in internet service. For example, a transient network failure may make the application server unreachable, an upgrade of the database software may conflict with the configuration of the application server, etc.

Chen et al. proposed an automated approach to diagnose failures in such systems using decision trees\(^5\). For each request, along with the result (success or failure), more detailed information are recorded as well, such as the software version number, host name, and database name. The logging system is called Centralized Application Logging (CAL) framework, which is a central repository of application-level logs. The log information of each request is asynchronously written to the Harvester cluster over persistent TCP connections. The goal is to induce the root causes of the known failures based on the request log data.

The key step of learning a decision tree is to decide, at a given node, which feature and testing value to choose to split
the data. Splits are chosen based on maximum of Gain function, until no further split is possible or necessary. The predicted causes of a request failure are the nodes along with the decision path in the decision tree, ranked by the failure counts.

5.3 Monitoring Performance Problem Using TAN

Cohen et al. (2) extend this work to use the results of metric attribution as a signature for failures. That is, he suggests identifying failures using the list of metrics or indicator in Service Level Objectives (SLO) correlated with the failure. He evaluates several interesting uses of these signatures, including searching for related problems in the past or in other systems, and tying together failures over time to identify recurring problems.

They are able to give a list of metrics, such a high load on the database or drop in network traffic along a link, which are highly correlated with the an externally visible SLO violation. This work is most related to our fault localization work, except that they apply correlation techniques to find metrics related to the underlying problem, but without requiring any tracking of dependencies between components and end-user requests. At the same time, they do assume the existence of a simple fault detector, such as a check for SLA violations, and might have a more difficult time correlating metrics for rare or minor failures, such when a fault affects only a small number of end-user requests.

Computation used Tree-Augmented Naive Bayesian Network (TAN) with Guilty function. A positive value suggests that \( x_i \) is “guilty” for the SLO violation. For a given violation instance, more than one metric could be guilty, the larger Guilty(\( x_i \)) is, the more likely that metric \( x_i \) reveals a true cause of the performance problem.

6. Analysis of Framework Need

From study of approaches in references, so we can induce that these are four basic function which each approach have. The four basic function are : sensor, log data base, statistical analyzer, and ranking/conclusion.

Sensor : collect data for analysis. Data whish is collected such as network packet or system log, depend on approach used.

Log DB : as information repository which is collected from both user and sensor.

Statistical Analyzer : do analysis to data from sensor and user information/indicator. It does computation using different function in each approach.

Ranking/Conclusion : order the most relevant causes associated with failure problem of system. User/operator will do action to repair based on the ranking.

7. Common Framework for automated diagnosis

To diagnose failures, sensors are deployed inside the system at lower layers and smaller components. The framework is shown in Figure 1. The sensors monitor the system behavior and runtime properties, which could be arbitrary information that may help diagnose the application faults. For instance, they may include whether a component finishes with success or failure, or CPU/IO load information. Combining with the information of success/failure at the application layer, then analyzed, the user may be able to induce which sensors produce the most relevant data that may contributed to the failure.

The problem is formalized as follows. Let \( Y \) denote the random variable of the application level outcome status for a particular sample. Suppose there are \( n \) sensors, and let \( X = (X_1, \ldots, X_n) \) denote the random variables of the states measured by the \( n \) sensors, the values of which are correlated with \( Y \) and may suggest the root cause of the problem. In usual fault diagnosis cases, \( Y \) is binary, which indicate either a successful run (0) or a failed one (1) for the application. In a performance diagnosis problem, \( Y \) might be a continuous variable dictating the performance result, such as throughput or latency, or it can be simplified to a binary proposition: either satisfied or violated SLO.

The automated diagnosis problem is to reason which sensors are the most correlated ones to the application failures. The answer to the question of how exactly to find the suspicious sensors is different for different statistical models. However, it is common to fit the problem into machine learning by training a classifier. The classifier also comes with a statistical model that describe the relations between the output of the sensors and the application output, which is essential in localizing the suspicious sensors.

8. Conclusion

This paper proposed a common framework for automated diagnosis in distributed information systems that captures the similarity of three different diagnosis systems using machine learning approaches. Framework deploy sensors inside system to monitor the system behavior and runtime
properties that give information to help diagnose faults. Using statistical analyzer, user can infer which sensors indicate the most relevant data may contributed to the failure.

References


