DustDoctor: A Self-healing Sensor Data Collection System

Mohammad Maifi Hasan Khan*, Hossein Ahmadi*, Gulustan Dogan†, Kannan Govindan‡,
Raghu Ganti§, Theodore Brown†, Jiawei Han*, Prasant Mohapatra‡, Tarek Abdelzaher*
mmkhan2@illinois.edu, hahmadi2@illinois.edu, gulustand@gmail.com, gkannan@cs.ucdavis.edu,
rganti@us.ibm.com, tbrown@gc.cuny.edu, hanj@illinois.edu, prasant@cs.ucdavis.edu, zaher@illinois.edu
*University of Illinois, †The City University of New York,
‡University of California, Davis, §IBM T.J. Watson Research Center

ABSTRACT

This demonstration presents a tool, called *DustDoctor*, for troubleshooting sensor data fusion systems where data are combined from multiple heterogeneous sources to compute actionable information. Application examples include target detection, critical infrastructure monitoring, and participatory sensing. In such systems, the correctness of end results may become compromised for a variety of possible reasons, such as node malfunction, bugs, environmental conditions unfavorable to certain sensors, or assumption mismatches (such as use of incompatible units on different nodes of the same distributed computation). DustDoctor adapts algorithms borrowed from previous discriminative mining literature to analyze data fusion flow graphs, called provenance graphs, and isolate sources and conditions correlated with anomalous results. This information is subsequently used to isolate malfunctioning components or filter out erroneous reports. We demonstrate our approach on MicaZ motes, running a simple data collection application, where users are allowed to inject a variety of different emulated faults, leaving it to DustDoctor to find and isolate them to prevent contamination of fusion results.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging-Distributed Debugging

General Terms

Design, Reliability, Experimentation

Keywords

Multi-sensor fusion, Data fusion, Wireless sensor networks, Quality of information

1. INTRODUCTION

We demonstrate a data fusion troubleshooting tool, called DustDoctor, that builds on a series of successful results on software troubleshooting published by the authors in Sensys 2008 (DustMiner) [4] and other conferences [3, 2]. The authors' previous tool, DustMiner, was designed to diagnose root causes of non-reproducible bugs in distributed software systems. It leveraged the insight that such bugs often arise as a result of unexpected interactions between components. Given examples of execution traces where the bug manifests and examples where it does not, the tool used discriminative sequence mining to find sequences of events most likely responsible for failure. DustDoctor applies this type of analysis to data fusion systems. By focusing on a well-defined category of distributed systems (namely, data fusion), we are able to implement "self-healing" functionality as well. For example, once the root cause of the problem is identified, data from the problem node or nodes can simply be ignored in order not to contaminate fusion results. This is the first demonstration of Dustminer, and its descendent, DustDoctor, since their conception.

In order to reason about root causes of problems when observing only final results of data fusion, DustDoctor exploits data provenance information of reported observations. Data fusion systems typically collect provenance information and send it along with the data, such that recipients know where the data came from and who operated on it. This information is invaluable in attributing failures to the correct causes. DustDoctor also leverages the received sensor data itself. Individual nodes' sensor readings, when reported, could be useful for providing further context that sheds light on possible causes of an anomaly. For example, when a camera turns dark and fails to deliver a picture, it may be useful to know whether or not the nearby light sensor was indeed in the dark. The operation of the tool involves two steps, namely, (i) data labeling, (ii) diagnosis. We describe each of these steps below.

1.1 Data Labeling

In order to trace root causes of problems one must first recognize that a problem has occurred. Hence, DustDoctor requires a data labeling capability, whereby results of fusion can be assigned a category, such as "good" or "bad". There are multiple ways data labeling can be done. The most obvious is for the user to indicate that results do not seem credible or for the system to perform application-specific automated sanity checks. In the absence of an educated user guess or a sanity check, prior work [6, 5] described algorithms to assess quality of information that may be used for labeling. For purposes of this demonstration, we augment our data fusion system with several sanity checks. The injected faults will cause these checks to fail, indicating that something is wrong.

1.2 Diagnosis

Once labeling is done, the tool processes the provenance graphs of labeled results, as well as any received sensor measurements, to generate discriminative features. A discriminative feature, in this context, refers to a subgraph of data fusion nodes, that is correlated with the occurrence of "bad" labels, together with any conditions under which such correlation is observed (e.g., only when it is dark, or only at high temperatures). Different results may have been computed from data of different nodes. For example, in a target tracking scenario, only sensors near the target might be sending data. The set of such sensors changes when the target moves. Each node in the data fusion graph can have an arbitrary number of parameters, such as sensorID, manufacturer, and driver version. Any combination of these can be a contributor to problems. For example, it could be that a certain driver is not compatible with sensors by a certain manufacturer, causing problems whenever such a sensor reports data. Finally, environmental context reported by sensors can also be part of the problem.

The diagnostic problem reduces to one of comparing the provenance graphs of all "good" results to the provenance graphs of all "bad" results. In each graph, each node is annotated by both its parameter values and its reported sensor measurements, if any. The comparison searches for combinations of nodes and their parameters and measurements, that are primarily correlated with "bad" behavior. To simplify this search, we replace each node that has N parameters (including sensor measurements, if any) with N virtual nodes representing a single parameter each, as well as an extra virtual node with no parameters (i.e., a total of $N+1$ virtual nodes) as shown in Figure 1. We then consider the discriminative power of each virtual node individually in classifying good and bad behavior. This helps remove irrelevant parameters early in the process and makes the rule mining more tractable for large scale data fusion graphs. More specifically, the tool applies association rule mining [1] to identify all sequences of virtual nodes (in Fig 1-b) that most accurately correlate with bad results. Association rule mining [1] is a well established approach to identify such sequences. The set of found sequences is then coalesced to form a single discriminative graph. The graph simultaneously describes which nodes are (likely to be) responsible, as well as which conditions hold true at these nodes, when bad fusion results are observed. The identified nodes are then disconnected from the rest of the data fusion system, whenever these offending conditions are observed, hence preventing result contamination. For example, a sensor that is observed to malfunction when humidity exceeds some level is disconnected when this humidity level is observed.

2. DEMO DESCRIPTION

To highlight the capabilities of our tool, in this demonstration, we shall troubleshoot a small-scale deployment of a typical data collection application implemented on MicaZ motes. Each node reports its sensor readings such as temperature, light, and accelerometer values. Simple functions of these measurements are also constructed across the node set. Nodes can join and leave dynamically when powered on or off.

For demonstration purposes, we shall use two categories of nodes. The first category always reports correct readings and performs correct computations. The second category is programmed to exhibit different types of failures, including those that are conditioned on some environmental context. For example, some node may malfunction only when it is in the dark (light sensor does not detect light) or only when it is up-side-down (the polarity of the vertical accelerometer axis reading is reversed). We shall also emulate failures that occur when certain nodes are used together (e.g., two nodes that unbeknowingly use different measurement units whose results are averaged).

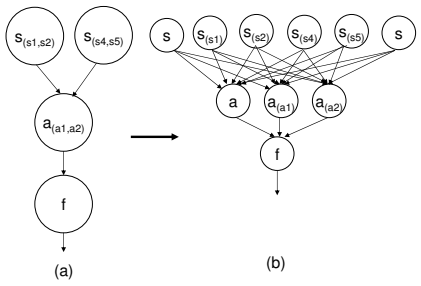

Figure 1: Handling of Multiple Parameters

During the demonstration, the user will be allowed to turn on or off both functional and faulty nodes to test our tool. When a faulty node is turned on and conditions that trigger the fault materialize, the sanity checks performed on fusion results will fail. This failure will be communicated to the user. With the user's permission, DustDoctor will troubleshoot the problem and learn the triggering condition that explains it. This condition will be explained to the user and, if so requested, will be subsequently utilized to filter out erroneous readings. The user will be able to repeat the experiments multiple times and test the tool with different kinds of faults of increasing complexity. A GUI will be provided for users to enhance the experience of interactive debugging.

3. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, pages 487–499, 1994.

[2] M. M. H. Khan, T. Abdelzaher, and K. K. Gupta. Towards diagnostic simulation in sensor networks. In *Proceedings of DCOSS*, 2008. Greece.

[3] M. M. H. Khan, T. Abdelzaher, J. Han, and H. Ahmadi. Finding symbolic bug patterns in sensor networks. In *Proceedings of DCOSS*, 2009. USA.

[4] M. M. H. Khan, H. K. Le, H. Ahmadi, T. F. Abdelzaher, and J. Han. Dustminer: troubleshooting interactive complexity bugs in sensor networks. In *Proceedings of SenSys*, pages 99–112, 2008.

[5] H.-S. Lim, Y.-S. Moon, and E. Bertino. Provenance-based trustworthiness assessment in sensor networks. In *DMSN*, pages 2–7, 2010.

[6] X. Wang, K. Govindan, and P. Mohapatra. Provenance based information trustworthiness evaluation in multi-hop networks. In *IEEE Globecom*, Florida, USA, December 2010.