Extended Abstract

Online Fingerpointing: Just-in-Time Problem Diagnosis for Distributed Systems

Keith Bare, advised by Priya Narasimhan

March 21, 2008

1 Overview

Distributed systems are growing both in terms of size and complexity. As a result, when a component fails, it can be difficult to determine which part of the system failed, let alone the cause of the failure. As the cost of downtime in production systems increases, it becomes economically important for problems to be discovered and repaired expediently.

Recent research has explored the creation of tools that attempt to automatically determine which component in a system failed. The resulting tools and algorithms are able to successfully implicate a faulty component in many situations. However, at this time, these tools see little, if any, use monitoring production systems.

A likely reason why systems administrators have not embraced tools for problem diagnosis is that existing tools cannot provide timely notification of problems. These tools collect data from the observed system continually, but only perform analysis a posteriori, after it was determined that a failure occurred by other means. Offline analysis, while useful for the evaluation of fingerpointing algorithms, is unlikely to excite operators in industry's data centers.

This thesis describes and evaluates FPT, a framework for online fingerpointing. FPT emphasizes flexibility while taking efforts to keep processing overheads low. The primary goal of this work is to explore the feasibility of just-in-time problem diagnosis. Additionally, it is hoped that FPT's flexibility will aid future research in problem diagnosis.

2 Implementation

The FPT framework can be used to construct fingerpointers that perform online analysis. It is highly customizable, so it can be used in many situations. A user provides a configuration file describing a directed acyclic graph with data processing modules at the vertices. FPT parses the configuration file, builds the specified graph, and then handles execution.

A graph structure is used for several reasons. First, it allows clean integration of data originating from a remote host in a distributed system. In essence, the fingerpointer will itself be distributed among the hosts, and edges between vertices on different hosts will be implemented with network connections. This also spreads computation among multiple machines, which will help keep computational overhead from becoming too high on any single machine. A second benefit is that a clear distinction between different parts of the fingerpointer facilitates evaluation of new modules. For example, a data source could be replaced with a different data source, while keeping the rest of a detector the same. Or, a new anomaly detection algorithm could be applied, just by replacing a single module. Finally, the graph structure can adapt in order to reduce the amount of computation required in the typical case. Parts of a detector can be dormant, except when some other part of the detector determines a threshold has been reached, and more data is necessary to accurately attribute a problem.

2.1 Configuration

An FPT configuration is specified in a format similar to a .INI file. A simple section describes a vertex in the graph, specifying the module to use, a unique identifier for the vertex, the edges to use as inputs, and module specific options. While this is enough to specify an arbitrary graph, specifying a complicated fingerpointer only using simple sections would certainly be onerous.

Configuration files can contain two additional section types that aim to simplify the construction of complicated graphs: $group sections and $foreach sections. A $group section creates and names a group of edges. Groups can be used as an input to a vertex (if the module supports input groups), or in conjunction with a $foreach section. Every vertex creates an automatic group of all its outputs identified by an '@' followed by the vertex's unique identifier. A $foreach section creates a vertex for each element in a group. The module to use for the vertices must be specified, as well as all the inputs the module requires. Additionally, a $foreach section can define groups containing all edges from the same output of each vertex the $foreach section created.

2.2 Data Processing Modules

Data processing modules are written as plug-ins. All modules interact with FPT using the same API. However, a module may take many forms of action. A module may introduce new data, process and forward data, or consume data.

The module API requires that plug-ins provide functions handling life cycle events and defining execution. Additionally, FPT provides many functions that a plug-in can call. These functions allow for inspection and manipulation of inputs and outputs, specification of scheduling criteria, and also provide access to values specified in the appropriate section of the configuration file.

3 Evaluation

[Note: since I have not yet completed evaluation of my system, this section currently describes my plans for evaluation. My thesis will present results.]

There are three important traits that will determine an online fingerpointer's success in providing just-in-time problem diagnosis: accurate implication of faulty system components, low overhead to prevent ill effects on the monitored system, and quick acknowledgment of anomalous situations. Experiments are designed to quantify these traits.

It is also useful to evaluate whether or not FPT is sufficiently flexible to serve as a platform for future work in problem diagnosis. While a tool's utility cannot be measured quantitatively, qualitative responses from other researchers can provide insight.

3.1 Experimental Setup

Hadoop is an open source implementation of MapReduce [4]. [I intend to provide more background information on Hadoop. Caveat: I'm currently planning on using Hadoop since it also being used by some other projects in Priya Narasimhan's research group. However, if initial experiences seem to indicate that working with Hadoop will be problematic, I may end up working with a different system.] Faults were injected into a single node in a cluster of machines [I will describe the machines] running Hadoop. An FPT fingerpointer is run on the nodes in the cluster, and its behavior is examined.

[I intend to describe the specifics of the Hadoop configuration, and the application being run on top of Hadoop. There will be justification why this is reasonable.]

[I intend to describe the faults that are injected, as well as the mechanism by which they are injected. The faults will probably include a crash, a hang, a memory leak, and packet loss.]

The FPT fingerpointer is configured to collect black box, OS-level performance metrics from the Linux /proc filesystem. Problem diagnosis is performed with some of the algorithms described in [6]. [I will state which ones I implement, and provide a very brief overview how they work. I will also include more specific details about the fingerpointer's structure.]

3.2 Accuracy

An inaccurate fingerpointing tool is unlikely to provide many benefits. False positives require administrators to waste effort fixing problems that do not actually exist, and false negatives may cause problems to go unnoticed.

Since FPT is configured to use previously evaluated algorithms, these experiments are mostly just make sure the number of false positives and false negatives remain similar to those in the previous work.

[I will determine what kind of experiments are best compared to previous work. Data collected from the experiments will be displayed in graphical and/or tabular form. If any of the data seems to disagree with previous results, potential causes for the deviations will be explored.]

3.3 Overhead

Overhead is measured by running the Hadoop workload without injecting faults. The run time will first be recorded when FPT is not running to provide a control data point. This can be compared

to run times when FPT is run with different configurations, in order to determine how much FPT's processing slows down the Hadoop computations.

[I will display the data collected, in graphical and/or tabular form, and make some comments.]

3.4 Detection Time

Detection time is measured by running a Hadoop workload with various faults injected. An FPT fingerpointer will monitor each run. Again, various algorithms and configurations will be used. In each case, the wall clock time when the fault is injected and the wall clock time when the fingerpointer determines there is a problem will be recorded.

[I will display the data collected, in graphical and/or tabular form, and make some comments.]

3.5 Responses

The FPT framework was presented to Priya Narasimhan's problem diagnosis research group. Initial responses indicate there is definite interest in the tool.

[Over the next month, the researchers plan to try using FPT. This will provide me with some good feedback on how other people feel about implementing data collection and analysis modules for FPT.]

4 Related Work

Architecturally, FPT seems to be most similar to EMERALD [7]. While FPT attempts to be a plug-in based tool for problem diagnosis, EMERALD is targeted toward intrusion detection. In some senses, problem diagnosis and intrusion detection are similar. However, the APIs defined by EMERALD are also more complicated than those defined by FPT.

Some existing work mentions running algorithms that can be used for problem diagnosis online. [3] mentions that the metric attribution algorithm was designed to have low computational overhead in order to allow the possibility of online execution. However, all results presented were analyzed offline. Magpie [1] can perform online clustering and extraction of request traces and from a running application. The request clusters can be analyzed to find anomalous behavior, but this analysis is not automated.

There are several existing approaches to offline problem diagnosis. FPT is designed to support approaches using statistical or machine learning methods to detect anomalous system states, similar to [2], [5], and [6]. FPT should be able to support similar methods, assuming that the computations carried out are fast enough to be computed in real time.

References

[1] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using Magpie for request extraction and workload modelling. *Symposium on Operating Systems Design and*

Implementation (San Francisco, CA, December 2004), pages 259–272. USENIX Association, 2004.

[2] Mike Y. Chen, Emre Kiciman, and Eric Brewer. Pinpoint: Problem Determination in Large, Dynamic Internet Services. *International Conference on Dependable Systems and Networks* (Washington, DC, 23–26 June 2002), pages 595–604. IEEE Computer Society, 2002.

[3] Ira Cohen, Steve Zhang, Moises Goldszmidt, Julie Symons, Terence Kelly, and Armando Fox. Capturing, indexing, clustering, and retrieving system history. *ACM Symposium on Operating System Principles* (Brighton, United Kingdom, 23–26 October 2005), pages 105–118. ACM, 2005.

[4] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pages 137-150, 2004.

[5] Emre Kiciman and Armando Fox. Detecting application-level failures in component-based internet services. Submitted for publication, September 2004.

[6] Soila Pertet, Rajeev Gandhi, and Priya Narasimhan. Fingerpointing correlated failures in replicated systems. *USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques* (Cambridge, MA, April 2007), 2007.

[7] Philip A. Porras and Peter G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. *20th NIST-NCSC National Information Systems Security Conference* (Baltimore, Maryland, 22–25 October 1997), pages 353–365. NIST-NCSC, 1997.