Synoptic : Unraveling Distributed Systems with Message-level Summarization and Inference Techniques

Slava Chernyak, Josh Goodwin, Sigurd Schneider, Ivan Beschastnikh

January 13, 2010

1 Introduction

Distributed systems are often difficult to debug and to understand. Some of this difficulty can be ascribed to the complex behavior of the nodes comprising the system. For example, nodes may act concurrently with one another and may participate in multiple protocols simultaneously. Another difficulty is the distributed nature of state. No node can observe all the system state, and due to asynchrony in the networking environment observations of remote state are stale unless protocols with significant overhead are used.

As a result, developers who design and implement distributed systems often gain confidence in their system by adhering to strict software engineering principles and methodologies such as test-driven development, and by amassing sufficient experimental evidence (e.g. message and node state traces) to indicate that the system operates without faults. Both of these methods are insufficient for constructing robust complex distributed systems and are problematic for two reasons.

First, it is typically impossible to explore all states of a complex distributed system. Therefore model checking, and testing are only practical for checking correctness of a subset of system modules in a simple setting. Second, it is challenging to understand the behavior of a distributed system from collected traces without performing some form of aggregation and reasoning over the traces.

1.1 Project Description

In our project we will design and implement *Synoptic* – a tool to analyze the network-level information generated by nodes in a distributed system and use inference and summarization techniques to help the user understand the behavior of the distributed system.

Usually the system developer is inundated with information about what each node in the system does. Our approach approaches the problem of understanding system behavior as a summarization and a data reduction challenge. We aim to mine patterns and relationships that range over the temporal properties of the captured system trace as well as the trace data itself. The resulting representation of trace must satisfy two contradictory goals:

TO WHAT END? WHAT TASKS, OR WHAT INFORMATION NEEDS?

- The representation must be as concise as possible
- The representation should retain as much information as possible

We foresee the research contribution of our project to be a characterization of a sweet spot between these two contradictory goals in the domain of representing executions of distributed systems. We also aim to define the metrics of conciseness and information in a way that reflects sound judgment of distributed system designers.

Note that unlike the problem of compression, which must retain *all* information, our representation can trade-off information content for conciseness. Additionally, unlike compressed content, our representation is intended to be inspected by a human.

I'M MISSING DETAILS HERE...

For this reason, the *form* of the representation is an important consideration.

Finally, although we employ inference and reason over the message trace, our representation of the distributed system is not a proof. We do not aim to generate system descriptions that hold in all circumstances. Instead we aim to generate a *useful* summary representation of the observed system behavior.

1.2 A Motivating Example

As an example, consider the `ping-pong` program in which $node_1$ periodically sends a ping message to $node_2$ and $node_2$ immediately replies with a pong message upon receiving the ping. The captured trace is a finite sequence of these messages.

Without considering the message data, the only possible summary of the observed behavior is that periodically $node_1$ sends some message to $node_2$, and that immediately after $node_2$ sends a message to $node_1$. Since message data is considered a black box, it is left out of the summary because its informational content is assumed to be 0.

ASSUMES A GLOBAL VIEW?

WRONG ANALOGY

If ~~we were also to know that~~ messages can be compared to one another then a simple analysis will reveal an alternating pattern in the trace. This pattern can be concisely summarized and must be presented to the user to avoid a loss of information from the trace.

WHAT TASK?

Finally, consider a `ping-pong` program in which every ping and pong message contains a sequence field that are independently incremented by 1 before message send by sender. In this case, a simple message comparison to find redundancy will reveal that the messages are in fact all unique. A more sophisticated approach is necessary. For example, the system can infer a relationship between the sequence fields of consecutive two ping or consecutive two pong messages. This relationship is simply $seq' = seq+1$. This representation captures all the message information content and is a concise representation of the message data.

WHAT DO YOU ANTICIPATE AS THE GRAMMAR OF PROPERTIES? THESE ARE REASONABLE EXAMPLES, BUT WHAT WILL THE FINAL SET LOOK LIKE?

ONLY OK IF NO 2 PINGS ARE EVER SIMULTANEOUSLY IN FLIGHT.

1.3 Assumptions

We make a few simplifying assumptions concerning the environment or the distributed system we analyze. First, we assume a non-broadcast networking medium at the message capture layer. For example, we do not handle wireless environments in which the nodes in the system sniff and receive all messages. However, we can operate in a wireless environment with all nodes restricted to using TCP for communication. Second, we assume that the set of nodes in the system is constant and does not change throughout the trace. That is, nodes do not fail and new nodes do not join the system.

GOOD, BUT, ALSO EXPLAIN THE REASON FOR THE RESTRICTIONS

2 Technical Approach

Synoptic is comprised of several components. First, all messages sent and received by the nodes of a distributed system are captured. This data is then preprocessed, i.e. reduced to the features of interest and content with 0 information is removed. Next, Synoptic infers properties and relationships between messages in the trace that lead to a more concise representation. As part of this step, the trace may be partitioned and different representations may be generated for different partitions of the trace. Moreover, information content may be elided and excluded to make the representation more concise. However, all such actions are meticulously recorded. Finally, the derived representation is presented to the user, along with all instances where information loss might have occurred.

2.1 Message Capture

We use the term *message* in a general sense to refer to a formatted message sent between two nodes in a distributed system. We assume that all messages have a source address, a destination address, and a data payload that may or may not be used in the analysis.

TIMESTAMP?

There are several ways to capture messages. For example, all traffic on a local machine could be sniffed (e.g. using `tcpdump`), or the system could be modified to use a custom network library that logs all messages sent and received to persistent storage. Initially we will explicitly capture messages by modifying the distributed systems that we study. OK.

To perform analysis on message payload, we will use protobuf [?] specifications of protocol messages exchanged between nodes in the system. This specification captures all the message fields and their types. The user will annotate the protobuf specifications of the system to indicate which fields should be used by Synoptic as part of its analysis.

2.2 Small and Interesting Distributed Systems

We plan to drive the development of Synoptic by applying it to interesting networked test program. Here is a set of such programs that we think are appropriate for this purpose:

- A `ping-pong` system as in the motivating example above. This system will consist of two nodes each oscillating between two states. It will be the job of our tool to infer these state sequences.

- A storage and retrieval system. The goal of our tool here may be to infer the constraint of a retrieve request always returning a modified result after a store request.

THESE SEEM QUITE SIMPLE. IT'S A REASONABLE PLACE TO START, BUT WHAT ARE THE LIMITS OR END GOALS?

- Other ideas include more complicated client-server systems where concurrent client access may force the inference engine to consider how to represent synchronization and concurrency.

2.3 Capturing the notion of Time

If the system is composed of just one physical host the local clock can be used for timestamping all messages. To analyze the behavior of multiple network nodes, a consistent notion of time is necessary. To start with, we will assume correctly synchronized clocks (e.g. using NTP). Later, we hope to establish a global partial ordering on the messages in the system with vector clock algorithms [?].

MUCH TOO BRIEF. I WANT YOU TO THINK MUCH MORE ABOUT CONCRETE SCENARIOS WHERE THIS WOULD BE USEFUL - THOSE MUST BE COMPELLING - AND HOW YOU WILL CONVINCE A SKEPTICAL REVIEWER.

2.4 Performing Inference

To perform the inference, we will use a predefined set of rules that our system with test against the collected trace. This method will be similar to Daikon [?]. We hope to then improve this approach to automatically generated logical and temporal properties from the trace and over message data.

FLATTERING TO THE INSTRUCTOR, BUT IS IT THE RIGHT APPROACH? DAIKON IS MOSTLY USELESS FOR TEMPORAL PROPERTIES, WHERE YOU WANT FSM OR GRAMMAR INFERENCE. AN EARLY EXAMPLE IS PERACOTTA FROM U.VA.

3 Evaluation

To evaluate our work we will run the program on selected distributed systems, including Hadoop [?] and Harmony [?]. The evaluation criteria of the representation outputed by Synoptic properties will include conciseness information content, and usefulness.

4 Timeline

- **January 25** : Related work review done
- **January 28** : Algorithm finalized
- **January 30** : Evaluation plan done – includes test programs, metrics, and reasoning behind selection
- **February 3** : Initial version of report due [intro, algorithm details, related work, evaluation plan]
- **February 6** : Test programs done
- **February 16** : Hadoop and Harmony instrumentation done
- **February 24** : Report with partial results due [running end-end prototype implementation that can generate some results]
- **March 10** : Experimentation complete
- **March 12** : Final report due
- **March 15** : (week of) Presentation