Visualizing Distributed State

Zipeng Liu
zipeng@cs.ubc.ca

Stewart Grant
sgrant09@cs.ubc.ca

1 Introduction and Motivation

Developing distributed systems is a difficult task. Inherent concurrency, and non-determinism complicate understanding how a system behaves. Developers lack tools for providing insight about the state of a system during it's execution. The lack of insight makes triaging bugs an arduous task involving the manual inspection of multiple logs. Visualization is useful for quickly articulating information. Currently there are no tools which visualize the state of a distributed systems to aid developers. Here we propose Dviz, a tool for visualizing the state of arbitrary distributed systems. Dviz profiles a systems execution by capturing a systems state and rendering features of its state transitions. Dviz produces predicable patters for systems executing under normal conditions, and irregular patterns for unusual executions. Developers can use irregularities in Dviz's output to quickly triage buggy behaviour.

2 Background

Distributed Snapshot is an algorithm which proposes that consistent distributed state can be captured without interfering with the execution of a system itself [1]. Distributed snapshots can be computed online or mined from a log containing vector clocks which provide a partial ordering of events in a system [8]. We consider distributed snapshots to be a fundamental granularity for examining consistent state. Our state analysis technique is therefore applied at the level of a distributed snapshot.

Dinv is a tool which detects likely data invariants in distributed systems [5]. Dinv operates by instrumenting distributed systems to log system state and vector clocks. Execution logs from the nodes of the system are merged together, and the state of the system is reconstructed and output as a distributed system trace. Dinv leverages Daikon to automatically infer data invariants on the trace. We plan to use Dinv as a tool for capturing distributed state.

Visualization is useful for helping users comprehend complicated data. Understanding distributed systems is a challenging task for system designers, software developers and students because of their inherent complexity. Prior work on the visualization on system traces has demonstrated that similarities in the traces between software versions can be meaningfully conveyed to developers [9]. Alternative work has demonstrated that visualizing concurrent system traces using partial orderings can help developers reason about interleaving executions [7] [6]. We consider the state of a system to be a direct artifact of a system trace. To our knowledge no attempts have been made to visualize distributed system traces, and by extension distributed state traces.

3 Proposed Approach

Our goal is to visualize distributed state in a comprehensive way, which provides developers with an intuition about faulty behaviour. To capture distributed state we will leverage Dinv, a tool for detecting likely data invariants in distributed systems [5].

Distributed state is inherently complex. In its raw form it consists of partially ordered instances of variable values. Logs of these variables can be massive. In order to convey distributed state to a developer, it must be summarized. We propose that the behaviour of a distributed system under correct execution follows predicable patters. Further that these patters are reflected the systems state. The first contribution that our project seeks to make is a state transition function $diff$ which measures the difference between distributed states. Our goal is to identify a $diff$ function which is sensitive to state transitions, and is normalized during repetitious behaviour.

Our initial approach is to measure state transitions as the XOR difference between variables in separate instances of distributed state. We formalize the state σ of a system to be a vector of variables $V = \{v_1, v_2, \ldots, v_n\}$. The XOR difference between two states σ_i and σ_j is $\text{XOR}(V_i, V_j)$. The resulting vector contains an indices for the XOR difference between each variable. The vector itself is n dimensional representation of the XOR velocity of the systems state. To measure the effectiveness of

our $diff$ function, we will initially plot its output as a simple line graph and check if predictable repetitions patters emerge between executions. Furthermore, we plan to plot the first and second order derivatives of $diff$, $diff'$ and $diff''$ respectively to determine if they also exhibit predictable patters. Initially we will try to detect patterns in Ricart-Agrawala. If predictable patters emerge we will introduce a bug which causes the algorithm to violate mutual exclusion. If predictable patterns emerge we will expand our research into larger more general system. If the difference is not noticeable but patterns emerge we will attempt to revise our visualization to better reflect the property. If no alternative visualizations demonstrate the irregularity, we will refine our $diff$ function to be more specific to Ricart-Agrawala's state variables.

4 Evaluation

To evaluate Dviz we will conduct a user study in which participants will identify buggy distributed systems. The purpose of the study will be to demonstrate that Dviz's output is useful for detecting buggy executions. Participants will be shown multiple visualization generated from the executions of three systems. To demonstrate Dviz's generality the system will be composed of a cross section of popular distributed applications. Specifically the systems will be etcd Raft [2], Groupcache [4] and Hashicorp Swim [3]. In all cases bugs will be introduced into the systems and the buggy visualization will be displayed alongside the non-faulty executions. Etcd Raft will be modified so that leader elections will result in more than one leader. Groupcache will not adhere to its key ownership policy. Swim will not fully propagate event messages to it's cluster. Study participants will be given a brief description of the systems and will be asked to select the visualization which they suspect to be faulty. Upon selecting a visualization participants will be asked to identify which aspects of the visual prompted their choice. Finally, they will be shown snippets of source code from the buggy execution, and be asked to identify the block of code responsible for the irregular visualization.

5 Timeline

- **Sept 25 - Oct 15** Write and revise proposal
- **Oct 16 - 22** Implement Prototype
- **Oct 23 - Nov 5** Test on various systems and revise design
- **Nov 6 - 20** Collect results and organize user study
- **Nov 20 - Dec 4** Perform user study
- **Dec 5 - 15** Write project report and prepare presentation
- **TBD** Present project and submit report

References

[1] K. M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, Feb. 1985.

[2] Coreos. Distributed reliable key-value store for the most critical data of a distributed system. https://github.com/coreos/etcd, 2013.

[3] A. Das, I. Gupta, and A. Motivala. Swim: Scalable weakly-consistent infection-style process group membership protocol. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 303–312. IEEE, 2002.

[4] B. Fitzpatrick. groupcache is a caching and cache-filling library, intended as a replacement for memcached in many cases. https://github.com/golang/groupcache, 2014.

[5] C. H. Grant Stewart and B. Ivan. Dinv: distrubted invariant dector. https://bitbucket.org/bestchai/dinv, 2016.

[6] S. Hahn, M. Trapp, N. Wuttke, and J. Dllner. Thread city: Combined visualization of structure and activity for the exploration of multi-threaded software systems. In *2015 19th International Conference on Information Visualisation*, pages 101–106, July 2015.

[7] B. Karran, J. Trmper, and J. Dllner. Synctrace: Visual thread-interplay analysis. In *Software Visualization (VISSOFT), 2013 First IEEE Working Conference on*, pages 1–10, Sept 2013.

[8] F. Mattern. Virtual Time and Global States of Distributed Systems. In *Parallel and Distributed Algorithms*, pages 215–226, 1989.

[9] J. Trmper, J. Dllner, and A. Telea. Multiscale visual comparison of execution traces. In *2013 21st International Conference on Program Comprehension (ICPC)*, pages 53–62, May 2013.