Visualizing Distributed State

Jodi Spacek
jodi.spacek@gmail.com

Stewart Grant
sgrant09@cs.ubc.ca

1 Introduction and Motivation

Developing and maintaining distributed systems is a difficult task due to the inherent complexity of concurrency and non-determinism. These factors complicate our comprehension of how a distributed system behaves. Developers lack a range of tools to provide insight about the state of a system during its execution. This lack of insight makes triaging bugs an arduous task because it creates the need to manually inspect multiple, dispersed server logs. Visualization is useful for quickly articulating information. Dviz is a visualization tool for distributed systems. Dviz uses logs of state and time to generate an approximate FSM mined from a system execution. FSM states are generated by differentiating the state and plotting these states on a plane. This provides a clustering of similar states. FSM states are also linked via a time curve which connects all of the states linearly. Users inspect states by examining concrete variable values at individual points along the curve. This implementation only approximates an FSM and does not provide a query interface for the rich data, such as data invariants which are readily available. We propose that Dviz's utility can be greatly improved upon by adding a flexible query language and by partitioning the time curve into a labeled FSM. These extensions to DViz facilitate user interaction and aggregate time curve information in a meaningful way.

2 Background

Distributed Snapshot algorithm proposes that consistent distributed state can be captured without interfering with the execution of a system itself [2]. Distributed snapshots can be computed online or mined from a log containing vector clocks. Mining provides a partial ordering of events in a system [8]. We consider distributed snapshots as 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 that 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.

Dviz is a visualization tool that plots distributed snapshots mined by Dinv onto a 2D plane. The position of each snapshot is determined by a XOR distance function. The distance between all states are computed then each state is plotted so that a the triangular inequality holds for all states. t-SNE clustering is used to compute snaphot position [7]. Each point is linked by a time curve [1]. Each point is also color coded based on their temporal ordering. The first snapshot is colored bright red and the final snapshot is dark brown. All intermediate points are colored with linear interpolated luminosity. Figure 1 is a Dviz plot generated from the execution of the etcd key value store [3]. The plot was generated from a test execution of 50 put requests followed by 50 get requests. Put requests compose the bright red cluster, while the dark brown corresponds to Get requests. Initial and final states are encoded by encircling them with blue. In its current implementation, Dviz has no facilities for automatically labeling clusters, making their significance a mystery to users. Further salient information about the similarities and differences of clusters is only attainable through the manual inspection of individual states. Labeling each cluster with cluster specific information, and allowing users to query features of the graph helps to contextualize the visualization.

3 Proposed Approach

Our goal is to improve Dviz by extending it. In this section we propose a labeling strategy for clusters, and transitions between clusters. Additionally we propose a query interface for exposing cluster, and inter cluster information to users.

FSM: Dviz's time curve is composed of clusters of states

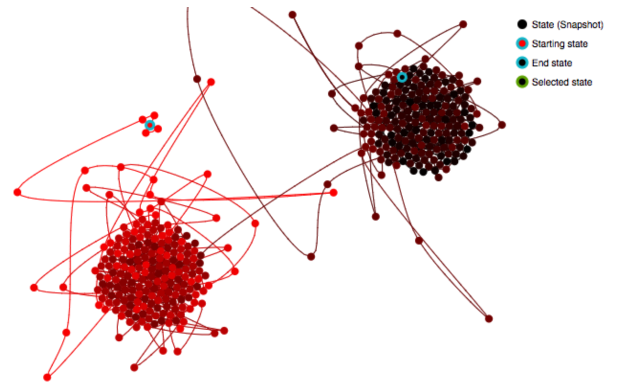

Figure 1: Time curve, with 2 clusters, generated from an etcd raft cluster processing 50 put requests, followed by 50 get requests.

and temporal transitions between states. Clusters form de facto macro states, such as in Figure 1, in which the bright red cluster represents the ”system is servicing put requests” state. Clusters, and cluster transitions form a partial FSM of an execution. A complete FSM includes labels for each cluster, and state transition.

Separating Clusters: In order to separate visual clusters into logical clusters we propose the use of k-means clustering [6]. Each individual state is identified only by it’s distance to each other state. K-means buckets based on a minimization of distance between states. K-means does not take into account the temporal relation between states, and can potentially be influenced by outliers. If k-means proves to have poor clustering we will experiment with density based clustering such as DBSCAN [4]. To measure the quality of either clustering technique we will complain them to manually annotated executions where the state of the system is known.

Labeling Clusters: Distributed snapshots contain variable values for each machine in the cluster. Distributed data invariants of each cluster can be automatically inferred using Dinv. To generate labels for each of the clusters we propose the use of Dinv’s invariants. Lables for each cluster will consist of of the set of invariants which are unique to a given cluster. Dinv infers a large number of invariants, this approach has the advantage of producing unique labels, and reducing the visual clutter generated by redundant labeling.

Labeling Edges: Edges between clusters also require labels to encode state transitions. The difference of invariants between clusters can be used to label the state transition. Using invariants to encode the transition has the advantage of keeping users in the context of data invariants, while clearly delineating between states.

Querying Data Invariants: Without substantial knowledge of a system, and the properties of it’s execution it is hard to determine what caused a cluster to appear, disappear, or move. Our goal is to answer why and why not questions about the time curve [9]. We propose the use of two forms of interactive queries to achieve this goal. For example a user may ask the question ”why does this cluster exist?”. The answer to this question is potentially complex; a complete justification requires information about distributed dataflow, and the values of individual variables. A simpler approximate answer can be given by examining invariants which hold on the cluster and identifying them throughout the execution. For example if on one cluster the invariant on two integer variables A and B, was $A > B$, and on all others $A < B$ held, a likely answer for the clusters existence is the invariant violation. This property can be encoded visually by popping out points in the time curve where the casual invariant was violated. To cause the states to pop out, the luminosity of all other points will be decreased while the query result will have its luminosity increased.

In addition to ”why, and why not” questions we propose supporting ”where” queries. Users can supply queries of the form ”where does **X** hold true”, where **X** is an invariant. Such queries will be answered using the aforementioned pop out technique.

4 Scenario of Use

Step 1

User views the clustering of system execution states and notes and anomaly with an extraneous end state.

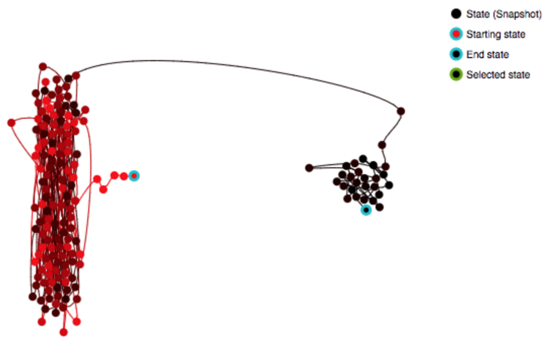

Figure 2: Time curve, with 2 clusters, generated from an etcd raft cluster processing 50 put requests, followed by 50 get requests.

Step 2 In Figure 3, the user enters a query to show the leader status where it is equal to true.

Step 3

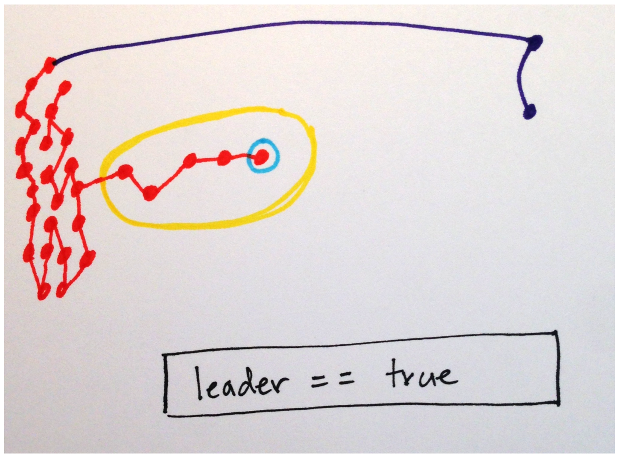

Figure 3: User selects the extraneous points and inputs a query.

The user can see that once the leader status is filtered, the points that do not belong are removed.

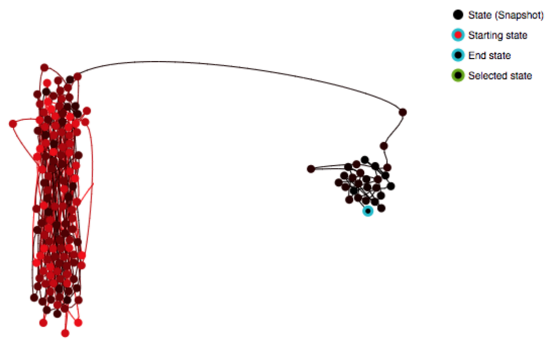

Figure 4: Time curve, with 2 clusters, withe filter applied.

5 Timeline

- **March 6** Write and revise proposal
- **March 13** Run clustering experiments and choose algorithm
- **March 20** Build support for cluster invariant analysis
- **March 27** Develop FSM labeling
- **April 3** Develop query engine and popout
- **April 10** Compose a user study
- **April 17** Start writing report, running study
- **April 24** Compile evaluation results
- **April 28** Submit report

6 Developer Skills

He we (Stewart Grant, and Jodi Spacek) overview our domain skills, in first person.

6.1 Stewart's Skills

For the past 2 years I have been a member of the NSS lab working with Ivan Beschastnikh. I work primarily at the intersection of software engineering and distributed systems. During my research I have built Dinv a distributed system specification miner, and other tools for understanding and testing distributed systems. In the fall of 2016 I took Ivan's distributed systems course and worked with Zipeng Liu on Dviz. Most of my work was conceptual and computational although I did have a hand in some of the visual components of Dviz. All of my Infoviz training has been built up by working with Zipeng, and taking 547 this semester.

6.2 Jodi's Skills

I've been working with Ivan Beschastnikh for the past year, as well as for a directed studies project during my undergrad. My interests lie in programming languages and distributed systems. During my 10+ year stint as a developer in industry, I learned about useful and not so useful development tools. One of the systems I helped build at Hootsuite was a tracing visualization tool for their microservices infrastructure. I'm particularly motivated in building tools for software engineers to help them comprehend their distributed systems.

References

[1] B. Bach, C. Shi, N. Heulot, T. Madhyastha, T. Grabowski, and P. Dragicevic. Time curves: Folding time to visualize patterns of temporal evolution in data. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2015.

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

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

[4] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.

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

[6] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[7] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

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

[9] B. A. Myers, D. A. Weitzman, A. J. Ko, and D. H. Chau. Answering why and why not questions in user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 397–406, New York, NY, USA, 2006. ACM.