DViz: Visualizing Distributed State

Jodi Spacek

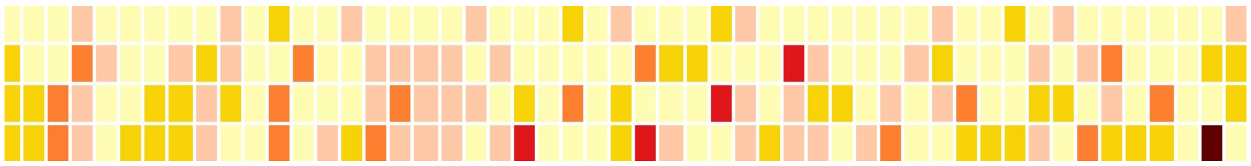


Fig. 1. Changes in state over a system execution

Abstract— This paper describes a tool that developers can use to visualize state changes for executions of software deployed in their distributed systems. The goal of this work is to provide a high level view of a system execution by highlighting changes which give developers clues about where to look in more finely grained event logs. A scenario is proposed for the goal of providing an interaction solution where a developer can investigate changes in the state of a system execution using the DViz tool.

Index Terms—Distributed systems, scatterplot, colourmap

1 Introduction

Distributed Systems are becoming more and more prevalent in industry with the advent of cloud platforms like Amazon AWS, Google Cloud, and Microsoft Azure that make it easy for an organization of any size to house their information in a distributed fashion. The distribution of data helps to ease the burden of computation on a single machine as well as removing a single point of failure.

However, these benefits come with drawbacks such as maintaining consistency of data between machines in a system. It is challenging to trace an execution in a distributed system because there is no absolute time to rely on, since servers may be located in different regions in the world and it is computationally heavy to synchronize to the millisecond. Developers working with a distributed system find themselves investigating several different machines in a network when the system behaves abnormally. In a non-distributed system, a developer could use a debugger to trace an execution in software and view the state of the system by setting breakpoints. The complexity of asynchronous communication in multiple instances of a software application deployed in the cloud makes debugging complex.

My project aims to help developers to visualize the state of their distributed system over a system execution. It facilitates this in two ways; 1) by providing a colour map of state changes, and 2) by logically connecting this colour map to an existing cluster scatterplot and event log originally built as part of Pangaea.

2 Related Work

2.1 Tensor Flow

Tensorflow provides a data visualization toolkit, TensorBoard, to visualize graphs generated from data flow graphs [1]. While this is geared towards machine learning and deep neural networks, the tool can be generalized to suit other types of systems. The graphing visualization can handle thousands of nodes by collapsing them into a scope. The grouping of nodes by name scopes contributes to the legibility of the resulting model. This system relies on the intervention of the developer to effectively group nodes by their name scopes, and does not infer the name scopes.

• *Jodi Spacek is with UBC. E-mail: jodispacek@gmail.com.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

2.2 Dapper

Dapper is a tracing tool developed at Google that instruments its software to generate trace information [10]. This system can track a system execution behind the scenes without requiring the developer to add anything to their programs. It also provides an API along with a web-based UI that developers can use to query their trace records. However, this tool is application specific as it is closely tied to the code base at Google.

2.3 Shiviz

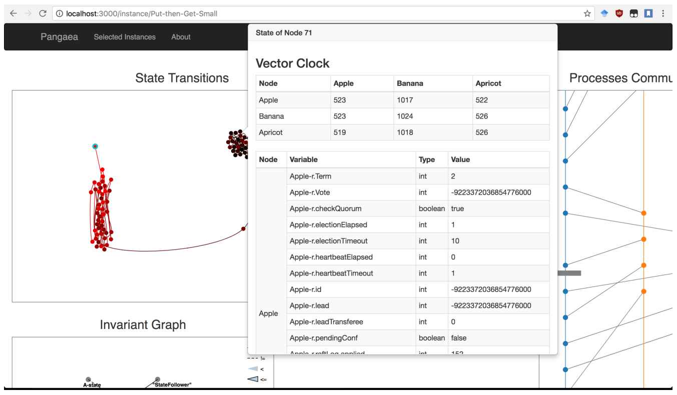
ShiViz is a visualization tool that sits on top of ShiVector, a mechanism that adds vector timestamps to distributed system logs [7]. ShiViz provides space-time diagrams of events that occur in a distributed system using partial ordering information [2]. This finely-grained tool gives a useful visualization for debugging log data but provides no inference of state in a system.

2.4 Pangaea

Pangaea is a tool for visualizing invariants in a distributed system execution [9]. It was developed by two graduate students at UBC as part of their research in distributed systems. This tool gives the developer two main views; 1) a process communication graph that shows communication between nodes, and 2) a cluster scatterplot of state invariants that correspond to snapshots of a system execution. This tool does an excellent job of presenting large amounts of data to the user, however, it leaves the responsibility of the interpretation of state changes to the developer. It also suffers from a high degree of occlusion due to its large popups which obscure most of the screen when viewed.

2.5 Hootsuite

Hootsuite is a social network management company that is built from small, independent pieces of code known as microservices. Microservices are small and plentiful which makes debugging non trivial [6]. Hootsuite has over 100 instances of 20 microservices deployed in their system which is hosted by Amazon AWS. During their Hootsuite hackathon, a team of developers attempted to create a tracing system similar to Google Dapper. They aimed to leverage their ELK stack, in particular to use Kibana logs and Graphite visualization tools to abstract a view of the system [5]. However, it was difficult to customize graphite and to interpose on all of the messages sent in the system because their microservices aren't as uniform as Google's. In addition, the visualization tool could not handle the load from all of the logs and



State of Node 71

Vector Clock

Node	Apple	Banana	Apricot
Apple	523	1017	522
Banana	523	1024	526
Apricot	519	1018	526

Node	Variable	Type	Value
Apple	Apple-r.Term	int	2
	Apple-r.Vote	int	-9223372036854776000
	Apple-r.checkQuorum	boolean	true
	Apple-r.electionElapsed	int	1
	Apple-r.electionTimeout	int	10
	Apple-r.heartbeatElapsed	int	0
	Apple-r.heartbeatTimeout	int	1
	Apple-r.id	int	-9223372036854776000
	Apple-r.lead	int	-9223372036854776000
	Apple-r.leadTransferee	int	0
	Apple-r.pendingConf	boolean	false

Fig. 2. A sample of Pangaea occlusion.

Table 1. DViz: Data Abstraction

Attribute Name	Type	Cardinality	Description
cut	categorical	1 - 100	Snapshot of system
node	categorical	3 - 5	Unique name for node
variables	assorted	50 - 100	state of the system
previous state	assorted	50 - 100	Stored link
clusters	categorical	Unique index	≤ 100

periodically caused the logging system, Kibana, to crash and remain unavailable for hours in production. Therefore, the project was backed out and the company is still searching for a solution. Developers manually coalesce logs from multiple nodes to trace executions.

3 Data and Task Abstractions

Using Visualization Analysis and Design as a guide, I will describe the developer drill down task [8]. Following this is the data abstraction that is represented in the tool.

3.1 Tasks

The main task that a developer can perform in DViz is to investigate changes in state over a system execution. State in this context refers to the values of variables that are spread across multiple nodes in a system. Nodes may also be referred to as hosts, but can more generally be thought of as servers that host instances of software programs.

A system execution is a sequence of events that are logically grouped and recorded for analysis over a set period of time. A snapshot is a slice over all the the nodes at a moment in time. This snapshot occurs when there are no messages in-flight during a period of inactivity in a system.

A developer finds a change in the system execution by viewing an abstraction of the changed variables across the servers. Using this information as a clue, the developer can gather more context around this snapshot in the clustering view. The clustering view provides more information on all of the variables at this point in the system execution. From the snapshot in the clustering view, the developer can navigate to the event log which includes more finely grained information in the process communication view. This process communication view includes all of the sends and receives between the nodes in the system when the snapshot was recorded.

3.2 Data

The data in this system is parsed from JSon files that are generated by an external program, Dinv [3]. DViz takes this data and uses it to generate the process communication log, the timecurve clustering view, and the changes matrix. Table 1 outlines the absolute and derived data used in the visualization.

Table 2. DViz: Marks and Channels

Attribute	Channel
Variable	x-axis
Snapshot state	y-axis
Degree of Change	Hue, Luminosity

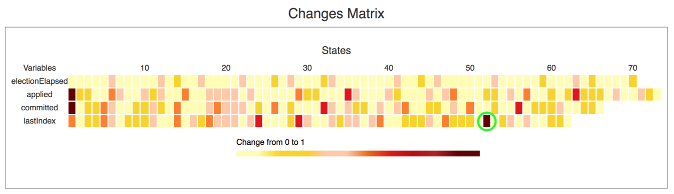


Fig. 3. A change matrix with a highlighted green cell of significant change.

The JSon data is static and is stored in an external web server. It is retrieved using an API call that returns a stored JSon file. This data has been manipulated by the DInv algorithms to generate unique invariants for the variable states which are in turn used in the cluster visualization. The raw data is also sent with the static JSon and it is this data that is used to detect changes in state.

The JSon file represents all of the data sent and received during a system execution. The top level of the JSon file is a cut, also referred to as a snapshot. These snapshots are ordered using vector timestamps that have been added to a software program. The data generally contains up to 100 snapshots per JSon file. Each cut contains the set of nodes that communicate with each other during the system execution. Each node has its own independent set of unique variable values. Data is organized per node in the raw data.

The data is filtered by changed values from an older snapshot to a more recent one. This filtering is performed before the data is presented to the developer thus it requires no interaction on the user's part. Table 3 outlines the channels used to display the variables once they have been filtered for changes.

4 Solution: Changes Matrix

The changes matrix solution provides a way for developers to better navigate the clusters and event logs in Pangaea. It describes, at a glance, volatility in a system execution which can be used to investigate the cluster scatterplot. Cells in the table correspond to the degree of change for a variable across all the nodes at a particular snapshot. Hue and saturation changes from a light yellow to a dark red to indicate a higher degree of changes in the normalized calculation.

4.1 Derived Data: Changes

Changes are detected across nodes per each snapshot. For example, if only one node has a change in a variable value, the change will still appear in the changes matrix. This is done because it is generally important for all nodes to have the same value for their variables, which is known as consistency. There are some special cases that are exceptions to this rule, eg. a leader is elected in a system and only one node should have a leader flag set to true. However, in this case we would want this leader change to be visualized because it is an important event in leader elections.

The sum of the variable values in each node were compared against the sum of the variables in the previous snapshot. Once the change is calculated, the value is normalized from 0 to 1 using feature scaling [11]. A maximum X value was stored for the variables and used in the calculation below. The minimum was set to 0 because all of the variables were larger than zero.

$$X' = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

Table 3. DViz: Analysis

System	Changes Matrix
What: Data	Sequential snapshot data points
What: Derived	Changes between current and previous snapshots
How: Reduce	Filter on changed values
How: Encode	Colour Map, Saturation and Hue
Why: Tasks	Locate, identify and compare changes in state

5 Implementation

The matrix was built using React components which render the HTML table. The cells of the table were connected to the change calculation. Additional javascript pages, html rendering methods, and CSS files were added for the matrix display. The scatterplot javascript code was modified to change the layout of the popup. The modified code can be found at a forked version of Pangaea [4].

5.1 Pangaea Changes

In addition to the changes matrix, a couple of improvements were made to the Pangaea interface. Two visual components of the Pangaea system were reworked to improve the user experience; the clustered scatterplot and the process communication graph.

The scatterplot and process communication components were moved closer together to use all of the available real estate. The size of the snapshot point popup in the scatterplot was minimized so that it would not occlude as much of the background. The background of the popup was made partially transparent so that the scatterplot and process communication components are more visible.

The layout of the data in the popup was rearranged so that is it aggregated by the variable rather than the node. This involved reworking the JSon data which is grouped by node. This reorganization facilitates side by side comparison of variable values at each node which gives the developer an easier view of the state changes.

5.2 Task Breakdown

In this project, my teammate Stewart and I started at opposite ends of the tool we proposed. My tasks involved the Javascript front end coding, as well as connecting to the Pangaea server. Stewart's code involved improving the algorithm that generates the cluster scatterplots. We were unable to meet in the middle in order to connect the Javascript to the new code. Due to time constraints, I decided to focus on improvements to the Pangaea system in order to provide something useful to the developer in the visual realm.

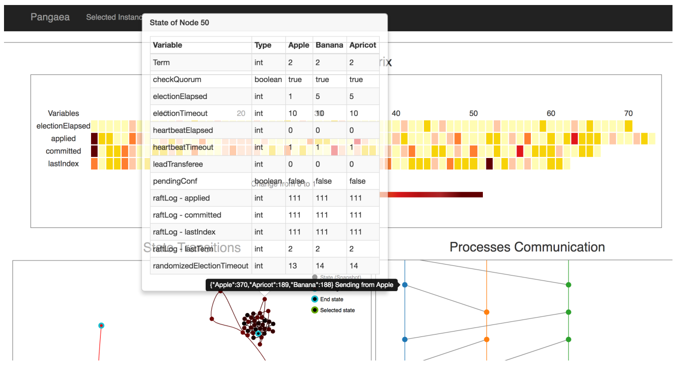
6 Results

Table 3 is an analysis of the changes matrix. It uses the raw JSon data that existed in Pangaea to derive normalized change data between two snapshots. This change data is used to filter the variables values so that only the changed values are displayed. This helps the user to identify changes in a system execution and it also provides a side by side comparison so that changes can be compared.

The developer uses information from the changes matrix to navigate to the relevant snapshot point. Clicking on this point in the cluster scatterplot gives the developer more information about the states of all of the nodes. A possible scenario begins with a developer who notices that one of the nodes is out of sync with the others in the system.

The user clicks the snapshot point in the cluster scatterplot that corresponds with the state number in the change matrix. The process communication changes to display the event in the graph that matches the snapshot point. The developer can hover over the node's vertexes to view the information that was passed from node to node. This finely-grained information reveals details on the behaviour of the system.

The user can follow the links between the snapshots in the cluster scatterplot to view how the system changes over time.


State of Node 50

Variable	Type	Apple	Banana	Apricot
Term	int	2	2	2
checkQuorum	boolean	true	true	true
electionElapsed	int	1	5	5
electionTimeout	int	10	10	10
heartbeatElapsed	int	0	0	0
heartbeatTimeout	int	1	1	1
leadTransferee	int	0	0	0
pendingConf	boolean	false	false	false
raftLog - applied	int	111	111	111
raftLog - committed	int	111	111	111
raftLog - lastIndex	int	111	111	111
raftLog - lastTerm	int	2	2	2
randomizedElectionTimeout	int	13	14	14

{"Apple":370,"Apricot":189,"Banana":188} Sending from Apple

Fig. 4. An example scenario where the user is viewing details of a snapshot.

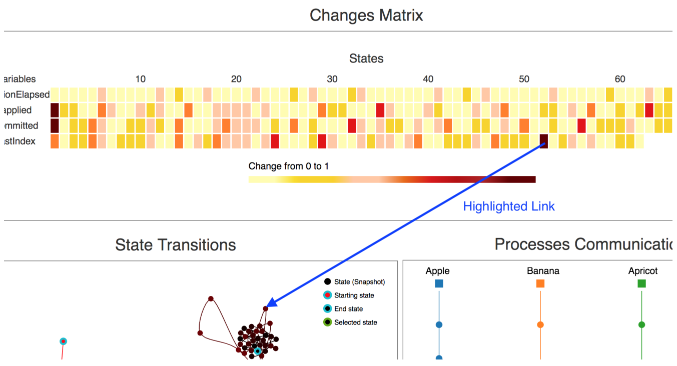


Fig. 5. A view of the change matrix where clicking on a cell links the user to the corresponding snapshot point.

7 Discussion and Future Work

A feature that would have been very helpful would be to implement linked highlighting between the changes matrix and cluster scatterplot. With this addition, the user would be able to easily view the point in the cluster scatterplot. Without this linkage, developers must guess where the correct snapshot point is in the scatterplot. Since the scatterplot has points that are on top of each other, points in the foreground occlude those in the background. It is possible that a point of interest is hidden behind some other point and the developer would not be able to drill down into the details.

Another addition would be to extend the change comparison across more than the previous point. The change value could incorporate more than one previous point's data by storing a cache of changed values. A weighted average would be a nice choice to show the degree of change over the entire systems' execution. A user could select multiple snapshot points for comparison.

8 Conclusion

This work provides an abstraction of change in a system execution to guide developers to areas of interest. The aim of the changes matrix is to ensure that the developer isn't asking questions like "How has my system changed?". Instead, this tool aims to help them to answer the question "Why did my system change?".

Minimal changes were made to the original Pangaea visualization tool that make a big impact on the usability of the tool. These changes remove occlusion and provide a view that is easy to compare changes in state.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] J. Abrahamson, I. Beschastnikh, Y. Brun, and M. D. Ernst. Shedding light on distributed system executions. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pp. 598–599. ACM, 2014.

[3] Dinv. `https://github.com/wantonsolutions/dinv-etcd`, 2017.

[4] Forked version of pangaea. `https://github.com/jspacek/Pangaea`, 2017.

[5] Elk stack 101. `http://code.hootsuite.com/elk-stack-101/`, 2017.

[6] Logging contextual info in an asynchronous scala application. `http://code.hootsuite.com/logging-contextual-info-in-an-asynchronous-scala-application`, 2017.

[7] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *CACM*, 21(7):558–565, 1978. doi: 10.1145/359545.359563

[8] T. Munzner. *Visualization analysis and design*. CRC Press, 2014.

[9] Pangaea. `https://github.com/zipengliu/Pangaea`, 2017.

[10] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. Technical report, Google, Inc., 2010.

[11] Normalization. `https://en.wikipedia.org/wiki/Normalization_(statistics)`, 2017.