iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 10 **Issue:** V **Month of publication:** May 2022

DOI: https://doi.org/10.22214/ijraset.2022.43001

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue V May 2022- Available at www.ijraset.com

VisCollab: Data Visualization with Provenance

Syed Mujtaba Jafri[1], Soumya Malgonde[2], Rohit Thakare[3], Bhavika Chuttar[4], Jakhete Sumitra[5]
[1, 2, 3, 4, 5]*Department of Information technology, Pune Institute of Computer Technology, Pune, India*

Abstract: Teams across the world are working together remotely today. Can we build a dashboard-style interface that uses some preliminary blockchain concepts to keep track of who did what on the dashboard? This can be used to keep track of user actions as well as for collaborative decision making.
Synchronous Collaboration: People collaborating on work at the same time. For example: Google Docs.
Asynchronous Collaboration: People collaborating on work, but making changes at different intervals. For example: Software Development (Coding). Some challenges with this include dealing with synchronous or asynchronous collaboration. For both synchronous and asynchronous, we would also need the ability to leave bookmarks/comments for interesting findings in the data for other team members to observe.
Keywords: Provenance, Synchronous collaboration, Asynchronous, collaboration, Data visualization.

I. INTRODUCTION

Teams across the world are working together remotely today. Can be built a dashboard-style interface that uses some preliminary blockchain concepts to keep track of who did what on the dashboard? This can be used to keep track of user actions as well as for collaborative decision making.

1) *Synchronous Collaboration:* People collaborating on work at the same time. For example: Google Docs.
2) *Asynchronous Collaboration:* People collaborating on work, but making changes at different intervals. For example: Software Development (Coding)

Some challenges with this include dealing with synchronous or asynchronous collaboration. For both synchronous and asynchronous, we would also need the ability to leave bookmarks/comments for interesting findings in the data for other team members to observe.

II. LITERATURE SURVEY

A. *Collaborative Visual Analytics Using Blockchain[1]*

The Concept: The paper tries to explore the viability of a blockchain back-end for collaborative visual analytics (C-VA) systems. Most literature available on this topic does not address the security and access issues related to the collaborators. This paper also explores the limits blockchain technology can have in C-VA systems.
The Implementation: The authors of the paper built Share.va; a framework that uses blockchain to log changes and insights derived through visual analytics dashboards.It enables users to share interactions, data filters, annotations, and notes. Every user interaction leads to a change in the dashboard state in which the framework logs to a back-end blockchain thus storing the provenance(origin/ownership) of an analysis. The blockchain system used was MultiChain.
Limitations: The blockchain based system has performance issues, it takes high response time for a block to be created and hence updates are slow. The authors mention that the application works well synchronously for about 6 users. It works okay with asynchronous collaboration as the requests are infrequent. Space and linearity issues are also there.

B. *VisConnect: Distributed Event Synchronization for Collaborative Visualization[2]*

1) *The Concept:* VisConnect is a web-based distributed collaborative visualization system. VisConnect replicates browser events, such as mousemove, across collaborators to produce synchronized visualizations. VisConnect supports simultaneous interaction with a lock system that gives each collaborator control over specific Document Object Model (DOM) elements. Users share a URL to the visualization that will connect their browsers in communication.
2) *The Implementation:* VisConnect is a peer-to-peer distributed system which synchronizes low-level pointer events, such as onclick and mousemove. It is written in TypeScript using Peer.js. This approach allows it to be easily added to many existing visualizations with only a few lines of code.
3) *Operation:* Load VisConnect -> Click the Icon -> Link Copied -> Send Link to Collaborator -> Collaborator Opens Link -> Connected Successfully.

III.TRRACK: A LIBRARY FOR PROVENANCE- TRACKING IN WEB-BASED VISUALIZATIONS[3]

The Concept: Trrack is a library to create and track provenance (history) in web-based apps. Trrack allows you to create and maintain a non-linear provenance graph representing the history of the state of your visualization. Through this graph, you can easily implement complete action recovery, as well as store custom metadata and annotations.
Trrack also allows for easy sharing of a visualization's current state through URL sharing. To share entire session history, Trrack allows for the import and exporting of provenance graphs, as well as has built in integration with firebase to store the graphs.
The Implementation:
Trrack uses a provenance graph approach where each recorded action results in a new node in the graph. State-based systems store the user-defined state of the application at every node, allowing for instant jumps to any node in the history.
Action based systems store the action required to get from one node to the next.
In Trrack, there is a different approach: differential states : a difference between the current node's state and the last node that stored the entire state.

Major pointers that were considered during the implementation were:

1) *Ease of Integration:* Trrack is designed to be framework agnostic. It can work with vanilla JavaScript, UI frameworks such as React, and state management libraries such as Redux.
2) *Sharing of State:* Trrack allows for easy sharing by sending the current URL to a collaborator.
3) *Reproducibility and Capturing Intent:* It is important to know the reason why a user was performing a certain action and that metadata was included by the authors in Trrack.
4) *Logging vs. Action Recovery:* Some actions developers want to track (e.g., for the purpose of logging) may be too frequent or inconsequential to include in the undo/redo chain. The concept of ephemeral nodes is introduced here.
5) *Data Characteristics:* The type of data that the applications have is mainly 2 dimensional.

IV.POLYCHROME: A CROSS-DEVICE FRAMEWORK FOR COLLABORATIVE WEB VISUALIZATION[4]

The Concept:
This paper contributes PolyChrome - a framework for building web-based collaborative visualizations for multi-surface environments. The framework supports:

1) co-browsing new web applications as well as legacy websites with no migration costs
2) An API to develop new web applications that can synchronize the UI state on multiple devices to support synchronous and asynchronous collaboration
3) Maintenance of state and input events on a server to handle common issues with distributed applications such as consistency management, conflict resolution, and undo operations.

Beyond the basic functionality to utilize implicit, explicit sharing models, and display space configuration through transformation to the global shared space, PolyChrome comes with a proxy server that supports collaborative use of legacy websites.

The Implementation: PolyChrome is implemented using HTML, JavaScript, and CSS. It is entirely cross-platform and works on any device with a modern web browser. PolyChrome consists of both client and server-side modules.
The server modules of PolyChrome have been built using Node.js, as the event-driven nature and non-blocking I/O of Node.js helps in efficiently managing interaction logs of multiple users over time. The client-side modules of PolyChrome contain the PolyChrome API that supports operation distribution and display space configuration.
It also has a proxy server that converts legacy web applications into collaborative applications, operation distribution, input, and rendering (visual representation) layers.
The framework interacts directly with the document object model (DOM) structure within the browser for display space configuration and also capturing browser events as operations.

PolyChrome consists of four modules:

- Operation (event) sharing
- Display space configuration
- Conflict, concurrency, and synchronization management
- A proxy server for serving legacy applications.

The design philosophy of PolyChrome is to treat the view of a web visualization at any time instance as a state that changes with user interaction. In essence, browser-level DOM events are the operations in PolyChrome applications. By capturing the user operations at the most atomic level on the browser in the form of DOM events, we can reconstruct the state of the visualization from the original rendering. The communication between clients of PolyChrome only happens in the form of event sharing.
PolyChrome uses PeerJS1 for P2P communication between the devices such as tabletops, tablets, and multi-screen displays, and communication via sockets from a client to server. The PolyChrome clients are connected over a peer-to-peer channel for event sharing, and they can also choose to connect to the server to store their interaction logs. This counters the major setback of using client-level web technologies, i.e., the lack of proper persistent storage of the interaction of a user when the webpage is closed.

V. VEGA-LITE: A GRAMMAR OF INTERACTIVE GRAPHICS[5]

The Concept: This paper contributes to Vega-Lite, a high-level grammar that enables rapid specification of interactive data visualizations. It combines a traditional grammar of graphics, providing visual encoding rules and a composition algebra for layered and multi-view displays, with a novel grammar of interaction.

The Implementation:
To support expressive interaction methods, authors contributed an algebra to compose singleview Vega-Lite specifications into multi-view displays using layer, concatenate, facet and repeat operators. Vega-Lite's compiler infers how input data should be reused across constituent views, and whether scale domains should be unioned or remain independent.
Grammar of graphics:
Unit specifications
View composition algebra [layer, concatenation, facet, repeat]
Nested views
Grammar of interaction: selection components and transforms

VI. OUR SOLUTION: VISCOLLAB

We leveraged the Trrack library in a NodeJS backend server to keep track of all changes and calculate provenance. For the frontend, we used D3.js and React. The frontend was connected to the backend using Websocket connections. The SocketIO library helped out with this. For authentication and authorization we used JWT tokens and persisted user data in a MongoDB cloud database. For deployment, we used the AWS platform for backend and Netlify for the frontend.
Changes made by clients in the frontend are sent to the backend via websocket emissions. It's the responsibility of the backend to relay these changes to all other clients. Along with this, Trrack also enables users to go back to a previous state. Going back to a previous state is also treated as a change and the same is relayed to all other clients. React reloads the visualization components every time it receives a change-event from the server.
As NodeJS is single threaded, there is no chance of race conditions. The change-event that reaches the server first gets the priority. The round trip of network calls has low latency thanks to websockets as they keep the TCP connections open unless one of the participants has a network outage.

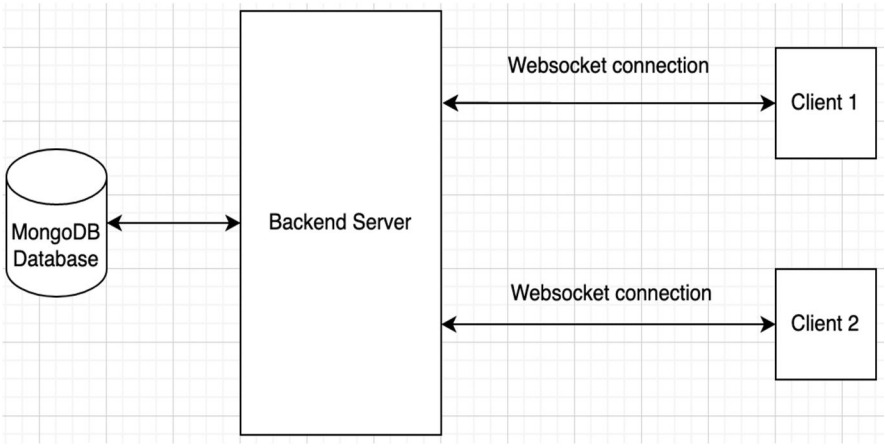


Fig 1: System architecture

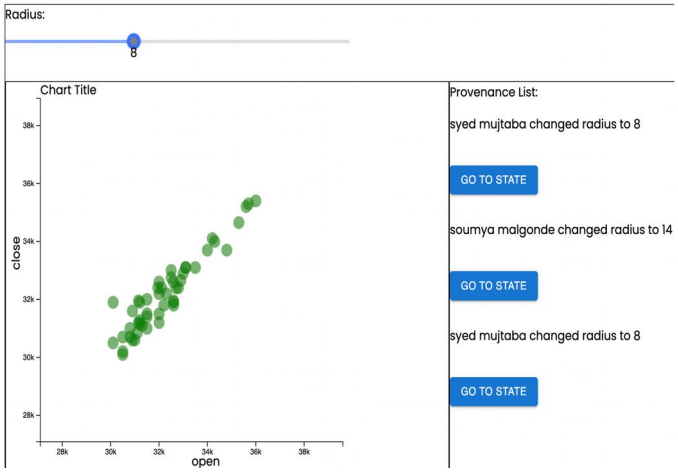


Fig 2: Slider visualization

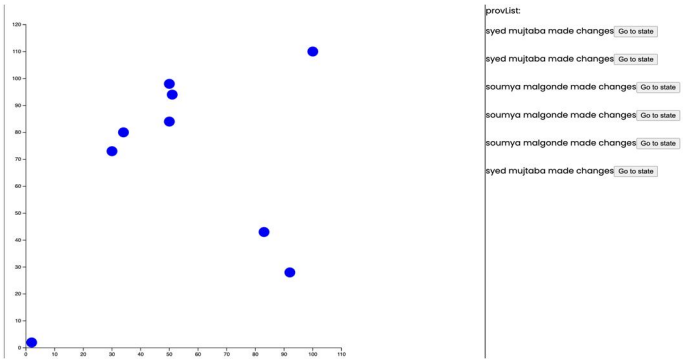


Fig 3: Scatterplot visualization

VII. CONCLUSIONS

We have designed VisCollab: a system for distributed collaborative visualization that requires little to no changes to existing visualization implementations, supports most web-based SVG data visualizations, scales to many collaborators with simultaneous interactions, and balances system safety with liveness for a smooth user experience.

One of the challenges we faced was using D3.js along with React. Both the libraries want full access to the DOM object. And the D3.js library assumes it has access to the DOM. This caused a lot of hiccups in the development process. Thus, we don't recommend using React and D3.js if someone wants to build a simple visualization with provenance tracking without much effort.

Through our use cases, we demonstrate the ease of implementation and use of VisCollab in multiple domains. We hope VisCollab will enable users to more effectively collaborate on visualizations; we encourage the visualization and human-computer interaction community to invest in distributed collaboration research by creating more collaborative applications and by studying interaction techniques and visualization designs specifically designed for collaborative use.

REFERENCES

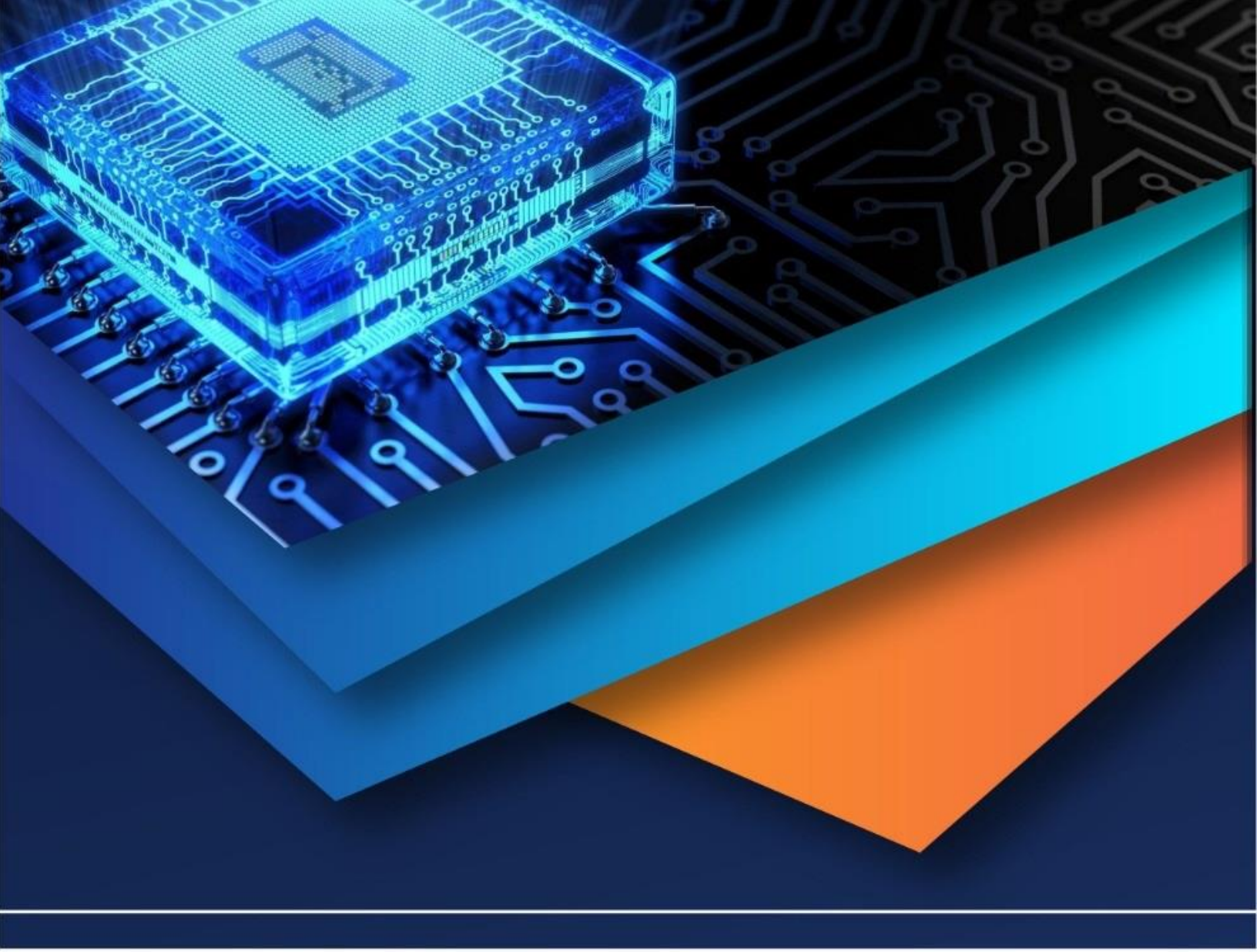
[1] Schwab, Michail, et al. "VisConnect: Distributed Event Synchronization for Collaborative Visualization." IEEE Transactions on Visualization and Computer Graphics 27.2 (2020): 347-357.

[2] Coelho, Darius, et al. "Collaborative Visual Analytics Using Blockchain." International Conference on Cooperative Design, Visualization and Engineering. Springer, Cham, 2020.

[3] Cutler, Zach, Kiran Gadhave, and Alexander Lex. "Trrack: A Library for Provenance-Tracking in Web-Based Visualizations." In 2020 IEEE Visualization Conference (VIS), pp. 116-120. IEEE, 2020 .

[4] Badam, Sriram Karthik, and Niklas Elmqvist. "Polychrome: A cross-device framework for collaborative web visualization." In Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces, pp.

[5] Satyanarayan, Arvind, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. "Vega-lite: A grammar of interactive graphics." IEEE transactions on visualization and computer graphics 23, no. 1 (2016): 341-350.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY