DebugAR: Mixed Dimensional Displays for Immersive Debugging of Distributed Systems

Patrick Reipschläger
Interactive Media Lab Dresden
Technische Universität Dresden
Dresden, Germany
patrick.reipschlaeger@tu-dresden.de

Stefan Gumhold
CGV lab
Technische Universität Dresden
Dresden, Germany
stefan.gumhold@tu-dresden.de

Burcu Kulahcioglu Ozkan
MPI SWS
Kaiserslautern, Germany
burcu@mpi-sws.org

Rupak Majumdar
MPI SWS
Kaiserslautern, Germany
rupak@mpi-sws.org

Aman Shankar Mathur
MPI SWS
Kaiserslautern, Germany
mathur@mpi-sws.org

Raimund Dachselt
Interactive Media Lab Dresden
Technische Universität Dresden
Dresden, Germany
dachselt@acm.org

CHI'18 Extended Abstracts, April 21–26, 2018, Montreal, QC, Canada
ACM 978-1-4503-5621-3/18/04.
https://doi.org/10.1145/3170427.3188679

Abstract
Distributed systems are very complex and in case of errors hard to debug. The high number of messages with non deterministic delivery timings, as well as message losses, data corruption and node crashes cannot be efficiently analyzed with traditional GUI tools. We propose to use immersive technologies in a multi-display environment to tackle these shortcomings. Our DebugAR approach shows a representation of the current systems state, message provenance, and the lifetime of participating nodes and offers layouting techniques. By providing a screen that shows a traditional text-log, we bridge the gap to conventional tools. Additionally, we propose an interactive 3D visualization of the message flow, combining an interactive tabletop with augmented reality using a head-mounted display. We are confident that our proposed solution can not only be used to analyze distributed system, but also for other time-dependent networks.

Author Keywords
Distributed Systems, Debugging, Augmented Reality, Interactive Surfaces, Multi-display Environments, 3D Visualizations

ACM Classification Keywords
H.5.2 [Information Interfaces and Presentation]: UI

Introduction

As cloud computing is getting widespread, distributed systems have become the common computing environment for many applications. Many popular services such as social media and networking applications, photo/video stores, and e-commerce sites operate on modern distributed systems. The design and analysis of distributed systems is much harder than for single systems, as they consist of a complex set of hardware and software components. While the system logic of a distributed system is highly encoded within the communication between components, high concurrency and asynchrony result in many different orderings of the in-transit messages. To design such distributed systems, a programmer needs to take these issues into account, as well as consider possible hardware failures [12]. Traditional 2D software tools are often not sufficient to support programmers in debugging such systems. Examples would be a bug caused by message ordering violations, which took a whole day to study [12], or a bug caused by a combination of several node crashes and reboots in Apache ZooKeeper [2].

We propose DebugAR, an easy-to-understand, yet powerful analysis tool for debugging distributed systems, using immersive technologies to expand the debugging capabilities of existing tools. By using a multi-display environment consisting of a text-log on a conventional desktop monitor and a sophisticated 3D visualization on an interactive surface, we bridge the gap to more traditional tools, which in contrast often utilize only a single screen. Furthermore, we use head-mounted Augmented Reality (AR) in conjunction with the interactive surface. This enables us to extend the output space of the displays to the third dimension, and to use natural multi-touch interaction to manipulate the visualization and explore vast datasets.

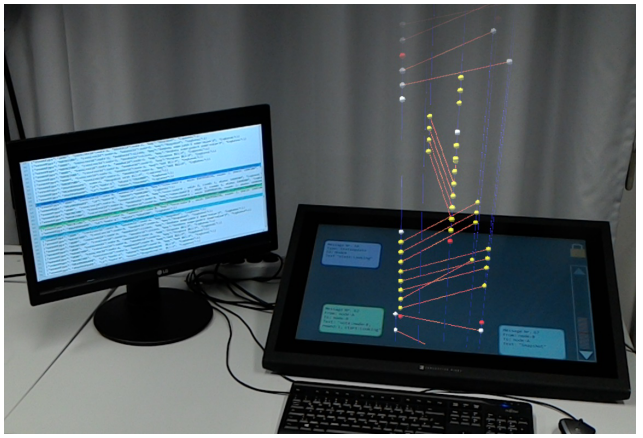

Figure 1: Setup of our proposed system, showing a conventional desktop monitor with a text-log on the left and an interactive surface with the AR 3D visualization on the right.

Related Work

Several tools were designed for the analysis and debugging of distributed systems [3, 6, 9]. Many of these tools involve some type of visualization to increase the comprehension of the system, which includes several distributed components and complex interactions between them. Typically, these tools extract some information statically from source code or dynamically from execution. They then process and display the information in a graphical form. Visualization of distributed system executions is investigated both for analyzing the performance [15] and the communication between the distributed nodes [1]. Most of these tools visualize the executions in a 2D environment. The increasing availability of 3D visualization frameworks and devices allows exploiting the third dimension for analysis. Recent work utilizes 3D for visualizing static information about the software, like Bonyuet et al. [4] or Codepark by Khaloo et al. [10]. From

Requirements for visualizing distributed systems

a) Representation of the current system state: A programmer needs to be immediately aware of which nodes are alive or in a crashed state, which node is the leader, which are followers, etc.

b) Configurable layout: The location of the nodes within the system are of significant concern for the programmer's mental map of the system.

c) Message provenance: It is important to be able to identifying problematic communication patterns, and track the causality of events within the system.

d) Aggregated views: In case of deep bugs the current state of nodes does not provide sufficient information, requiring an aggregated view of all crashes/reboots of nodes together with their corresponding messages.

the related work, we derived a set of requirements for a visualization to be a useful debugging tool for distributed systems (see left sidebar).

Our proposed concept of using multiple visualizations for the analysis is an example for a multi-display environment (MDE). MDEs consist of heterogeneous displays, which often do not constitute a continuous display space (with the exception of, e.g., BendDesk [14]), and which have been applied to various application areas, including visual analytics [5]. A more specialized area are distributed user interfaces, where components are distributed across one or more of the dimensions input, output, platform, space, and time [7]. There have also been examples of using Augmented Reality in combination with conventional displays as early as 1991 with *Hybrid User Interfaces* by Feiner and Shamash [8]. Most of these works are primarily concerned with distributing the user interface itself and do not focus on visualizations specifically. *Coordinated & multiple views* [13] for visualizations provide synchronizing interaction across these views by using techniques like linking and brushing. In contrast to MDEs, these systems however are mostly limited to non-distributed display setup, although there is work from Langner et al. [11] using multiple mobile devices. To the best of our knowledge, our tool is the first to utilize a hybrid immersive environment for the analysis and debugging of distributed systems.

Concepts

The essence of our DebugAR concept is to support the debugging of distributed system by using an immersive multi-display environment (MDE) to visualize the data flow and state of all involved actors. Our proposed MDE consists of two screens (see Figure 1): The first one is a conventional desktop monitor primarily showing a text-log of the messages between actors. It provides programmers with a familiar debugging-tool similar to traditional systems. The second screen is a tilted interactive multi-touch surface and shows a 3D visualization of the distributed systems execution traces over time. It is generated by the display in conjunction with Augmented Reality (AR), provided by a head-mounted display (HMD). In the following, we focus on the 3D visualization as a new, immersive form of debugging and how it aids the programmers in understanding the system behavior, as well as on the connection between both displays.

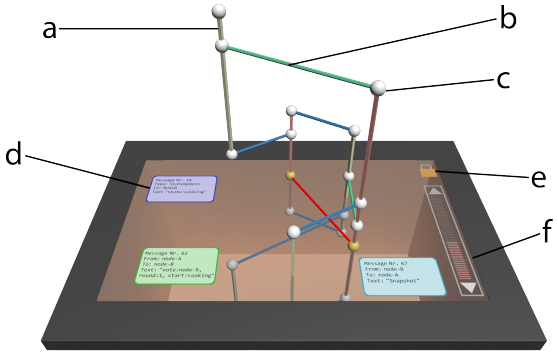

Figure 2: Detailed picture of AR 3D visualization showing: a) The actor lifelines, b) color coded messages between the actors, c) state indicators on actor lifelines, d) pinned messages, e) the lock symbol, and f) the slider widget.

3D Visualization The 3D visualization shows the development of the distributed system over time by visualizing node lifelines, their color-coded state at discreet time frames, and the sequential order of the messages passed between them. The current time-step is located directly on the screen of the interactive surface. Colored spheres on a nodes lifeline encode its current state at discreet time

steps (see Figure 2c), for example if a node has crashed or whether it has recovered again. Messages between nodes are displayed as animated color-coded lines between those spheres, indicating the type and direction of the message (see Figure 2b). This makes it easy for a programmer to perceive the current state of the system.

The space above the display is used to visualize previous time steps, while future ones are visualized behind the display. Depending on user preferences, a time step itself is defined either by a single message passed between two nodes, or by an aggregation of messages between discreet time steps. The number of time steps visible above and below the display can also be configured by the user. All other time steps beyond the adjustable limit are faded out and only become visible if the current time frame is modified accordingly.

The lifetime of nodes is visualized by a thick line perpendicular to the display screen (see Figure 2a). This line also serves as a representation of the node itself and continues through all the time steps in which a node was active. A user can choose if a colored sphere is placed on the lifeline at every discreet time step or only when the state of a node has changed. Regardless of this, the current frame on the display screen always shows the state of every node that was alive at that time. All parts of the 3D visualization are sole AR objects visualized through the HMD the user is wearing.

To interact with the visualization, the multi-touch capabilities of the interactive surface are used. A dedicated 2D slider widget (see Figure 2f) displayed on the surface modifies the current time frame using simple drag gestures on the widget, or by taping the buttons at top and bottom of the widget. The number of visible time steps is increased or decreased by performing a pinch gesture on the widget.

Inter-Display Connection The two displays, the one showing the text-log and the other which serves as the host for the 3D visualization, are intended to be used in conjunction with each other and not as stand alone devices. Brushing and linking is applied to highlight selected items on both visualizations. For instance, when a message is selected in the 3D visualization, it is also highlighted in the same color within the text log. Furthermore, the currently visible time frames of both visualizations can be linked together. This means that moving the current time frame in one visualization will also move the other one accordingly, keeping them synchronized and making it easier for a user to perceive the current state on both visualizations. The lock can be enabled or disabled any time by using the lock icon in the upper left corner of the interactive surface (see Figure 2e).

Message and Node Pinning To get a detailed description of a message or the current state of a node, a user can tap the corresponding element of the current time frame. An info box is then pinned on the interactive surface itself as a 2D object, not as an AR object. This makes use of the higher resolution of the display in comparison to the AR HMD, improving the readability (see Figure 2d). The info boxes can be freely organized by dragging them around the screen. Pinned objects remain on the display even when the current time frame is moved. This enables the user to create a collection of detailed information on messages and nodes that are of particular interest to him or her when tracking down a bug.

Layouting To further support a programmer in debugging of distributed systems, we provide several layouting mechanisms, which can be used to organize the actors based on different aspects. Our default layout aims to maximize the readability of the visualization by ordering all actors as a circle with equal spacing between each actor. Messages

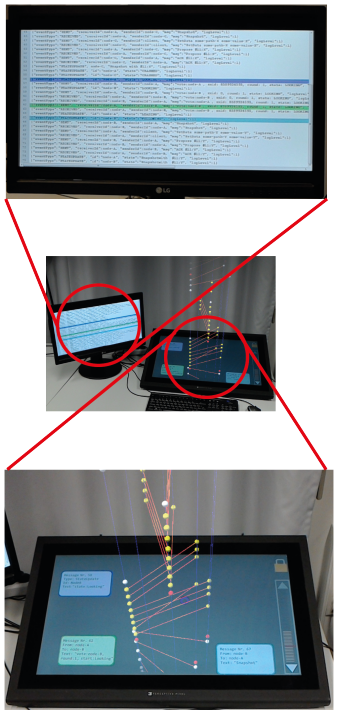

Figure 3: Snapshots of our current prototype showing the 3D visualization and interactive surface (Picture taken through Microsoft HoloLens).

between actors will pass through the center of the circle to minimize the risk of misinterpreting sender and recipient of a message. However, other than the message flow, this layout has no inherent expression of the relation of actors to each other. An alternative layout orders all actors based on the quantity of their communication with each other. Actors that send a lot of messages to each other are positioned close to each other, while actors with sparse communication are further away, resulting in the formation of several clusters. Additionally, actors can also be layouted by their physical location (if available), which for example enables the programmer to check if errors are related to the physical proximity of the actors. Furthermore, a user can also refine a layout by simply dragging a node in the current time frame to another location.

Implementation

We implemented an early prototype (see Figure 3) to demonstrate the main aspects of our concepts, using the Unity 3D engine for the 3D visualization and the MS HoloLens as AR HMD. Unity and WPF are used for the display clients. For the communication between all involved devices we use a client-server architecture, where all clients connect to a central server. Clients can send events to indicate interaction or state changes, and other clients can subscribe to events of specific types and get notifications. One challenge for our system is to synchronize the coordinate systems of the interactive surface and the HoloLens for the 3D visualization, to make sure AR objects appear correctly in relation to the display. We use AR markers, displayed on the interactive surface during an initial configuration step, which are detected by the HoloLens (using ARToolKit) and used to place an anchor/root for the 3D visualization.

Conclusion

We presented DebugAR, an immersive environment for debugging distributed systems. Our proposed concept provides a representation of the current system state through our Augmented Reality 3D visualization, combined with a horizontal multi-touch display, as well as several layouting techniques. Thus, it addresses some of the requirements for visualizations of distributed systems (see Requirements Sidebar). Wo do not yet support message provenance and aggregated views, be plan to add these capabilities in the future. While still at an early stage, we believe our solution has a lot of potential to be a useful tool in supporting programmers in their debugging-tasks. Verifying this claim of course requires conducting a user study, comparing our solution to current debugging environments. We are confident that the idea of combining vertical and horizontal desktop monitors with an Augmented Reality visualization is suitable not only for debugging distributed systems, but analyzing other types of time-dependent networks as well.

Acknowledgments

This work was partially funded by
BMBF Project No. 01IS14014 (ScaDS).

REFERENCES

1. Jenny Abrahamson, Ivan Beschastnikh, Yuriy Brun, and Michael D. Ernst. 2014. Shedding Light on Distributed System Executions. In *Proc. ICSE'14*. ACM, New York, NY, USA, 598–599. DOI: http://dx.doi.org/10.1145/2591062.2591134
2. Apache Issues. 2017. Zookeper-335. http://issues.apache.org/jira/browse/ZOOKEEPER-335. (2017). Accessed: 2017-12-11.
3. Elisa Gonzalez Boix, Carlos Noguera, Tom Van Cutsem, Wolfgang De Meuter, and Theo D'Hondt.

2011. REME-D: A Reflective Epidemic Message-oriented Debugger for Ambient-oriented Applications. In *Proc. SAC'11 (SAC '11)*. ACM, New York, NY, USA, 1275–1281. DOI: http://dx.doi.org/10.1145/1982185.1982463

4. D. Bonyuet, M. Ma, and K. Jaffrey. 2004. 3D visualization for software development. In *Proc. ICWS'04.* 708–715. DOI: http://dx.doi.org/10.1109/ICWS.2004.1314802

5. Lauren Bradel, Alex Endert, Kristen Koch, Christopher Andrews, and Chris North. 2013. Large High Resolution Displays for Co-located Collaborative Sensemaking: Display Usage and Territoriality. *Int. J. Hum.-Comput. Stud.* 71, 11 (Nov. 2013), 1078–1088. DOI: http://dx.doi.org/10.1016/j.ijhcs.2013.07.004

6. Darren Dao, Jeannie Albrecht, Charles Killian, and Amin Vahdat. 2009. Live Debugging of Distributed Systems. In *Proc. CC'09 (CC '09)*. Springer-Verlag, Berlin, Heidelberg, 94–108. DOI: http://dx.doi.org/10.1007/978-3-642-00722-4_8

7. Niklas Elmqvist. 2011. *Distributed User Interfaces: State of the Art*. Springer London, London, 1–12. DOI: http://dx.doi.org/10.1007/978-1-4471-2271-5_1

8. Steven Feiner and Ari Shamash. 1991. Hybrid User Interfaces: Breeding Virtually Bigger Interfaces for Physically Smaller Computers. In *Proc. UIST'91 (UIST '91)*. ACM, New York, NY, USA, 9–17. DOI: http://dx.doi.org/10.1145/120782.120783

9. Dennis Geels, Gautam Altekar, Scott Shenker, and Ion Stoica. 2006. Replay Debugging for Distributed Applications. In *Pro. ATEC'06 (ATEC '06)*. USENIX Association, Berkeley, CA, USA, 27–27. http://dl.acm.org/citation.cfm?id=1267359.1267386

10. Pooya Khaloo, Mehran Maghoumi, Eugene Taranta, David Bettner, and Joseph Laviola. 2017. Code Park: A New 3D Code Visualization Tool. In *Proc. VISSOFT'17*. IEEE, 43–53.

11. Ricardo Langner, Tom Horak, and Raimund Dachselt. 2017. VisTiles: Coordinating and Combining Co-located Mobile Devices for Visual Data Exploration. *IEEE Trans. Vis. Comput. Graph.* 24, no. 1 (10 2017), 11. http://dx.doi.org/10.1109/TVCG.2017.2744019

12. Tanakorn Leesatapornwongsa, Jeffrey F. Lukman, Shan Lu, and Haryadi S. Gunawi. 2016. TaxDC: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems. *SIGPLAN Not.* 51, 4 (March 2016), 517–530. DOI: http://dx.doi.org/10.1145/2954679.2872374

13. J. C. Roberts. 2007. State of the Art: Coordinated Multiple Views in Exploratory Visualization. In *Proc. CMV'07*. 61–71. DOI: http://dx.doi.org/10.1109/CMV.2007.20

14. Malte Weiss, Simon Voelker, Christine Sutter, and Jan Borchers. 2010. BendDesk: Dragging Across the Curve. In *Proc. ITS'10 (ITS '10)*. ACM, New York, NY, USA, 1–10. DOI: http://dx.doi.org/10.1145/1936652.1936654

15. C. Eric Wu, Anthony Bolmarcich, Marc Snir, David Wootton, Farid Parpia, Anthony Chan, Ewing Lusk, and William Gropp. 2000. From Trace Generation to Visualization: A Performance Framework for Distributed Parallel Systems. In *Proc. SC'00 (SC '00)*. IEEE Computer Society, Washington, DC, USA, Article 50. http://dl.acm.org/citation.cfm?id=370049.370458