Combining ShiViz and Simulation Suite for Distributed Algorithms

Michael Kowal

Abstract UBC's ShiViz and UNBC's Simulation Suite for Distributed Algorithms are both powerful tools used to assist with designing and improving distributed systems. By combing the perks of each of them, users will be able to do more without the need for expensive hardware and software solutions.

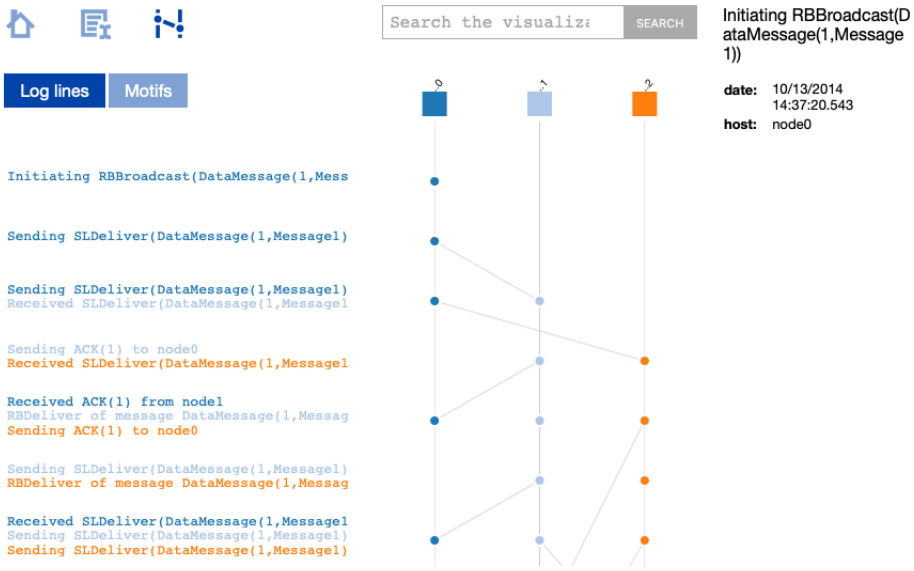
Introduction The purpose of this project was to combine two distributed systems applications and provide ways for users to easily use the modifications. All of the modifications are done to UNBC's Simulation Suite for Distributed Algorithms (SSDA), ShiViz was never changed, but libraries provided by the ShiViz team were used. The project took approximately one and a half months to complete, excluding the documentation.

Vector Clocks The primary relationship between the two programs is their vector clocks. Both of them track events based on a vector clock that records which node in a collection of nodes is currently preforming an action. The clock will increment for that node upon completion of that action. The two programs both do this in a slightly different way, requiring the use of two different clocks when generating a ShiViz log.

ShiViz ShiViz is a software for visualizing data flow through distributed systems. It was created by students at the University of British Columbia. The tool helps users understand the flow of a concurrent or distributed system by providing users with an interactive graph that displays where data travels and when. It allows users to see patterns, called "motifs" in the data flow as well as provides many different ways to search and manipulate the graph to see more specific events. The software

requires specially formatted log files that, when input into the site, get interpreted and displayed correctly as an interactive graph.

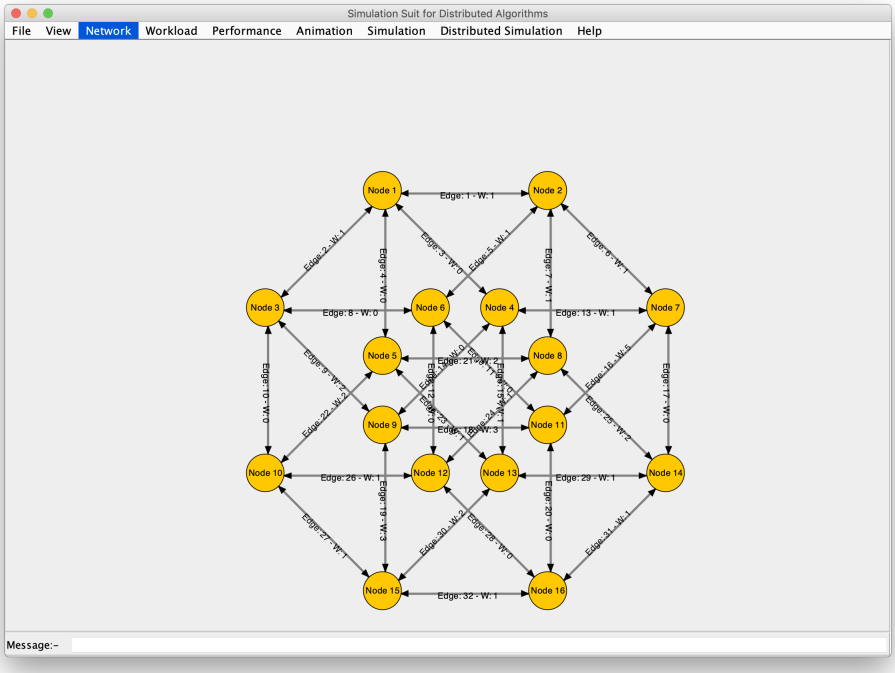
Figure 1: ShiViz example visualization



Simulation Suite for Distributed Software The primary software that was used in this project is the SSDA created by students at the University of Northern British Columbia. This software gives users the ability to design a distributed system and test that system without having to pay the huge costs of buying all the necessary hardware. The tool provides many different pre-made topologies that can be modified by the user as well as the option to draw an entire network from scratch. The user creates nodes and then assigns connections between those nodes. It is then possible to adjust the behaviour of each individual node as well as apply different settings to the entire network. There are other features of the app as well, like an animation that displays the flow of information from one node to another, as well as ...

Benefits of Integration Both of these softwares provide users with tools to visualize different aspects of distributed systems. These tools, when combined, give users an even wider array of resources with which to learn about distributed systems. By enabling the use of ShiViz's logs, users are able to see exactly where the flow of information between different nodes is. It adds a more in depth layer of information that allows the SSDA to have more applications for a wider user base.

Figure 2: SSDA user interface



Changes Made There are no changes made to the ShiViz software and libraries, but there are a number of small changes to the SSDA. The first additions made were adding the ShiViz vector clock library to the project. After that the JVec class that was provided by ShiViz was incorporated into the SSDA. To do that, a JVec object was created for each node and would record every time that a message is sent or received by that node.

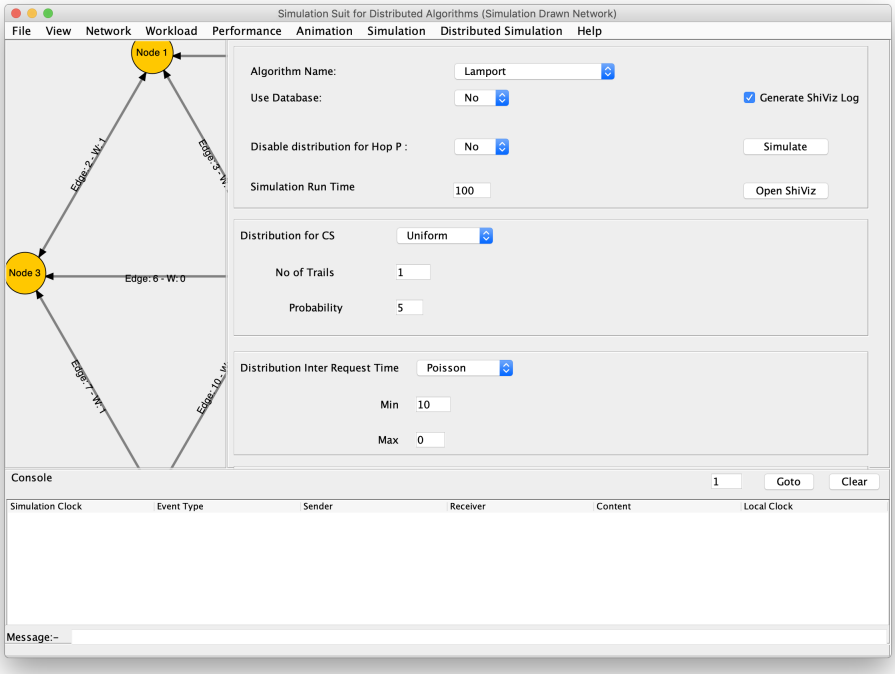
```
//sending a message
encodedMsg = node.getVCInfo().prepareSend("Node " + getNodeId() + ":
    Sending Message: " + message.getContent(),
    message.getContent().getBytes());
//receiving a message
node.getVCInfo().unpackReceive("Node " + getNodeId() + ": Receiving
    Message: " + messageRecieved.getContent() + ", from node " +
    messageRecieved.getFinalSender(), encodedMsg);
```

Once each node was being tracked by a JVec object, the GUI elements were changed

to allow users to create ShiViz logs from within the SSDA. Finally, a class was added that takes all of the individual nodes' logs and condenses them into one that included the regular expressions needed by the ShiViz software.

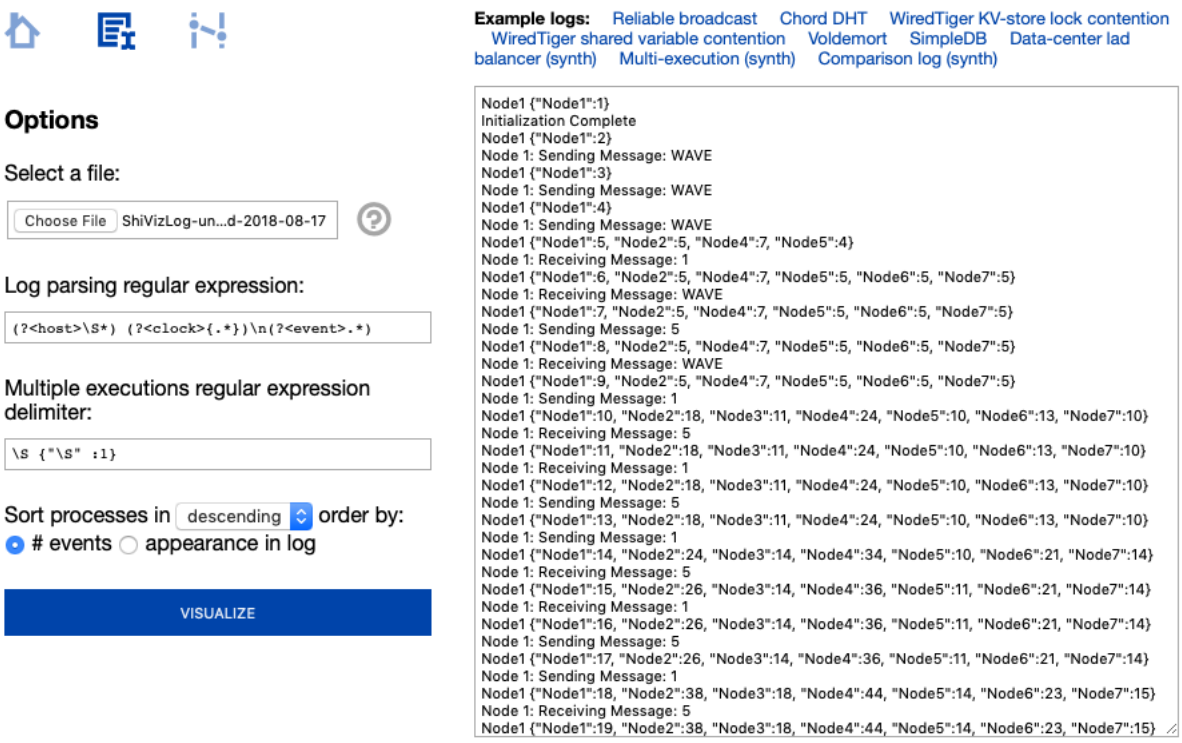
How to Use the Modified Software In order for a user to create a ShiViz compatible log from within the SSDA, they simply need to click the box labeled "Generate ShiViz Log" on the Simulation page of the SSDA. After setting up a simulation, it can be run by pressing the "Simulate" button. This will create a Log file in the "LogFiles" folder in the same directory as the SSDA. In order for the user to see ShiViz's visualization of the log file, they must open ShiViz in a browser. By clicking the "Open ShiViz" button in the simulation screen of the SSDA, ShiViz will open. Once there, clicking the "Try out ShiViz" button will bring them to a screen that will ask for a file. The appropriate file can be found in the LogFiles folder where

Figure 3: SSDA simulation panel



the log was originally created. The last step is to click "Visualize" and ShiViz will create a visualization of the selected log.

Figure 4: ShiViz file manipulator



Challenges Faced There were many challenges faced while completing this project. Much of the time early on was spent learning about the basics of distributed systems and what the two softwares could do. It took some time to fully understand what was needed. Once all of the research was complete, the next hurdle was understanding where to properly integrate the ShiViz vector clock library with the UNBC software. Creating the logs and recording each sent message was relatively straightforward, but properly recording the received messages proved to be quite the task. This took longer than anything else and required much more research into what exactly was happening in the library. Eventually, after much time and frustration, the solution was found by using the encoding provided upon message creation and decoding the message at the receiver's end. With that challenge complete, the remaining polish was not difficult. Adding UI elements and consolidating all of the individual logs was simple File IO.

Conclusion Stuff happened and it was interesting.