Liquid, Autonomous and Decentralized Stream Processing for the Web of Things

Masiar Babazadeh

Faculty of Informatics, University of Lugano (USI), Switzerland
{name.surname}@usi.ch

Abstract. In recent years we have witnessed the rise in number of smart devices and sensors connected through the Web. This led researchers to explore the World Wide Web as a platform to orchestrate such devices. In this demo we show how we are able to harmonize heterogeneous hardware for home automation systems with the Web Liquid Streams (WLS) framework. The WLS framework lets developers implement topologies of data streams across a heterogeneous pool of devices thanks to Node.JS and the Web browser. By using JavaScript, the lingua franca of the Web, we are able to write the stream operators once and run them anywhere a Web browser or Node.JS can run. The demo shows a home automation system application that can seamlessly run on different kind of devices.

Keywords: Web of Things, Streaming Applications, Liquid Software, Home Automation Systems

1 Introduction

The increase in the number of smart devices and sensors has led developers to explore the Web as a platform to organize and make these devices interoperate. The accessibility of such hardware has also led the rise of the Maker culture [1], a subculture that extends in a technological way the Do It Yourself (DIY) movement. Makers often have a basic understanding of electrical engineering concepts and programming paradigms, and are able to program their microcontrollers to automatise (part of) their houses. Given the lack of programming experience, integrating different kinds of microcontrollers through the Web may appear a rather difficult task for Makers, which either fragment their home automation in small set-ups, or have to deal with data streams and data synchronization in a heterogeneous environment.

In this demo paper we demonstrate the use of the Web Liquid Streams framework [2, 3] to help Makers build topologies of distributed data stream operators that are able to run on different kind of hardware: from a small microcontroller, passing to laptops and big Web servers.

WLS follows the Liquid software paradigm [4], by which applications can exploit computational resources from all the devices owned by the user, and dynamically migrate from one device to the other, for example in faulty scenarios.

This turns out to be very useful when applied to a distributed streaming application scenario, where the data stream is expected to be up and flowing for a very long time.

2 Framework

While building Web applications across Web enabled devices may nowadays appear a relatively simple task thanks to Node.JS, WebRTC and WebSockets, it is much more difficult to setup an infrastructure that is able to arbitrarily exploit the JavaScript Event Loop on the available hosting machines.

With the Web Liquid Streams framework, developers can create topologies of distributed streaming operators and run them across a pool of heterogeneous hardware resources. Thanks to JavaScript, topologies may be implemented using a single programming language and the WLS' primitives. The WLS runtime is then in charge to deploy the implemented operators on the available devices, connect them through sockets and start the data stream.

WLS is able to tolerate faults and changes in the computing environment at runtime, by migrating the execution of the operators on other available devices, and re-routing the topology accordingly. This is done by a control infrastructure which is also in charge of solving bottlenecks in the computation, by increasing or decreasing the allocated resources on the host.

WLS can be used by Makers that can program in JavaScript and want to automate their homes with Web-enabled sensors and microcontrollers. After developing the operators scripts, Makers have only to define a topology description file that tells the WLS runtime how the operators have to be wired. Makers do not have to worry about the actual communication channels, it is the runtime's task to create the right sockets to make two arbitrary machines communicate.

3 Demo

The demo focuses on integrating a streaming topology across a pool of heterogeneous devices, simulating a home environment. We make use of temperature, pressure, and humidity sensors as well as a microphone to monitor the environment in different rooms. The available hardware, that in WLS we call "peers", are two Raspberry Pis (first model), two Tessels (first model), one laptop and one smartphone with the Google Chrome Web browser installed, and two Web servers running Node.JS. We show how these peers can interoperate seamlessly thanks to our framework by just installing it, and writing both the script operators and the topology description file.

Figure 1 illustrates the demonstrated topology. We setup three producers, two running on Tessels (peer 0 and peer 1) and one on a Raspberry Pi (peer 2). The data gathered is sent both to a consumer (peer 3), running on a smart phone with the Chrome Web browser installed, and to a filter (peer 4) running on a Web server. The Web browser consumer receives the data from the sensors and displays the status of the microcontrollers. If one producer dies, the Web browser

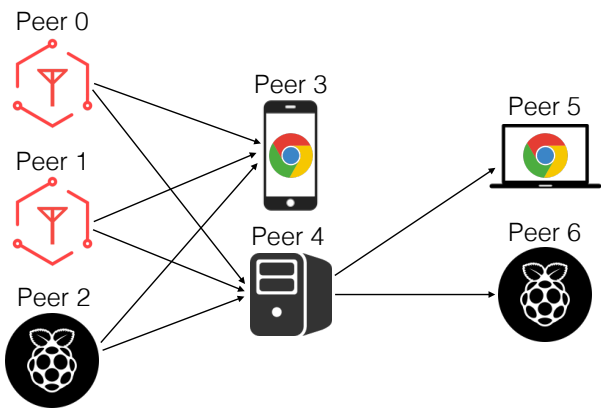

Fig. 1: The demonstrated topology.

will show it on the page. The Web server instead gathers the data, stores it into a database and computes a mean of the last received values. These measures will then be forwarded to both the consumers at the end of the topology. The Web browser consumer (peer 5) graphically shows the obtained measurements by the means of interactive graphs, while the Raspberry Pi consumer (peer 6) works as an actuator, turning a fan on if the temperature is too high, and lighting red alert lights if the noise level is too high.

Listing 1.1: Example filter script

```
1  var k = require('WLS.js');
2
3  //initialize operator state
4  k.createOperator(function(measurements) {
5      //store the measurements
6      k.db_store("temperatures",
7      {
8        "room_id" : measurements.id,
9        "temp" : measurements.temperatureData
10     });
11     //[...] store all the other measurements
12
13     //compute means of that room
14     var toBeSent = {};
15     toBeSent.tempMean = avg( k.db_get("temperatures",
           measurements.id, 1000) );
16     //[...] compute all the other averages
17
18     //send data downstream
19     k.send(toBeSent);
20 });
```

Listing 1.1 shows part of the filter script that is used in the demo. First our library is imported, then we declare the initialization of the operator and pass a callback function that is executed each time a message is received. We store the data, retrieve the mean and forward it downstream.

This demo also shows how effective our fault tolerance is, thanks to our controller infrastructure. During the demonstration, we will proceed to disconnect the first consumer running on the smart phone and show how the computation is migrated seamlessly to the laptop, effectively maintaining the structure and semantics of the topology. We will then proceed to connect another Web server and cause a fault in server hosting the filter operator. Also in this case, the computation will be migrated to the new available hosting server. The topology will be automatically rewired, and the data stream will restart.

Finally, we will update the producer scripts, increasing their sending rate. This will create a small bottleneck on the filter operator, which will be solved autonomously by the controller infrastructure by allocating more resources on the hosting peer, effectively parallelising the execution of the operator.

4 Conclusions

In this demo we have shown how the Web Liquid Streams framework can be used by Makers to develop their personal streaming topologies, passing through their microcontrollers, personal computers or home Web servers. Thanks to JavaScript, Makers only have to know JavaScript and install WLS across their devices. The WLS runtime will then take care of deploying the implemented operators on the available devices and start the data stream. Thanks to our controller interface, the topology becomes fault tolerant, being able to autonomously migrate the faulty operators on other available machines and rewiring the topology, maintaining its structure and semantics intact.

Acknowledgements The work is supported by the Hasler Foundation with the Liquid Software Architecture (LiSA) project of the SMARTWORLD initiative.

References

1. Anderson, C.: Makers : the new industrial revolution. Random House Business Books, London (2012)
2. Babazadeh, M., Gallidabino, A., Pautasso, C.: Decentralized stream processing over web-enabled devices. In: 4th European Conference on Service-Oriented and Cloud Computing. Volume 9306., Taormina, Italy, Springer (September 2015) 3–18
3. Babazadeh, M., Gallidabino, A., Pautasso, C.: Liquid stream processing across web browsers and web servers. In: 15th International Conference on Web Engineering (ICWE 2015), Rotterdam, NL, Springer (June 2015)
4. Mikkonnen, T., Systa, K., Pautasso, C.: Towards liquid web applications. In: 15th International Conference on Web Engineering (ICWE 2015), Rotterdam, NL, Springer (June 2015)