Quest Journals
Journal of Software Engineering and Simulation
Volume1 ~ Issue 3 (2013) pp: 01-06
ISSN(Online) :2321-3795 ISSN (Print):2321-3809
www.questjournals.org

Research Paper

Updating web content in real time using Node.js

Edlira Kalemi, Phd [#1], Krisel Tola [*2],

[#] *Computer Science Department, "Aleksander Moisiu" University of Durres Albania*

Received 29 October, 2013; **A**ccepted 30 November, 2013© The author(s) 2013. Published with open access at **www.questjournal.org**

ABSTRACT: *The real-time web is a set of technologies and practices that enable users to receive information as soon as it is published by its authors, rather than requiring that they or their software check a source periodically for updates. In order to achieve this, we can use different technologies. One of them is Node.js. Through this article we want to describe this technology and highlight the advantages of using Node.js as a technology to update web content in real time. To achieve this objective we have made some tests and measurements for the time spend to update information using these technology versus other technologies like Php and Apache.*

I. INTRODUCTION

Requirements for real-time applications are increasing day by day. The need to be updated throughout the time, regarding the interested information or data is becoming bigger. We are recently working on a project, related to stock trading, where the main goal is to update the stock data every second and show those data to the user through a web application. The big question here is what technology to apply to make the almost real-time update possible. After consulting different material and made some testing we decided to use the Node.js technology. Most of web developers are used to implement the JavaScript language only over the client side. Node.js as a platform gives us the opportunity to write JavaScript scripts for the server side too. So, having the same language on both client and server makes you think that things are running faster here. We have mostly compared Node.js with Php/ Apache. Node.js is a new technology and, comparing to other script languages like Php, has lack of support. With Node.js we can create our http server which will not rely on Apache server like Php does. For this test we will retrieve stock data for 10 different companies from Yahoo! Finance using both Node.js and Php / Apache. After retrieving the data we will show it to the client using html. Then we will analyse the execution time of the created applications on Load Impact, testing them with 50 simulated browser users for approximately 10 minutes. Both applications are hosted on c9.io which is a cloud service that supports Node.js and Php /Apache. In the above sections we will go into Node.js details and we will also explain the code used and the results achieved from the test.

II. BACKGROUND

"Node.js is a platform build on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices [1].

Why do we need scalable network applications?

Network scalability means a fast network server that can handle a considerable amount of users making requests to our server. This is very important because the real time data's other than overloading the cpu with the frequent requests towards the data provider will also be accesed by a high number of visitors. Sometimes the web application might be under different kind of attacks such as distributed attacks which opens a lot of false http connections. If the server is not scalable it will not be able to support those kind of attacks and consequently will not serve to our requests on time.

*Corresponding Author: Krisel Tola

Computer Science Department, "Aleksander Moisiu" University of Durres Albania

What should we understand with event-driven programming?

Event-driven programming (EDP) or event-based programming is a programming paradigm in which the flow of the program is determined by events [2]. In this kind of programming the server waits for an event to happen such as a button click, on focus, key press etc.. . When the server receives the event it fires the action related to event. In our case we create a loop on the client side which will fire an event every one second.

What is a non-blocking input/output model?

Asynchronous I/O, in computer science, is a form of input/output processing that permits other processing to continue before the transmission has finished [3]. This model gives us the opportunity to run other processes while the server is loading our stock data. These is really important in our case because we need to do other processes like buying or selling trades and in the meanwhile the stock data's must be updated regularly. So if we chose a synchronous model we will not give our users the possibility to make these actions because the server would be busy updating our data.

III. FRAMEWORKS

Like many other platforms, Node.js uses frameworks which makes our life easier. Express and Socket.io are widely applied in this kind of technology. With Express we can easily create a server, configure it and set the port the server will listen. We also use Express to specify the folder in which the files that will be served to the clients are. Express has the capability to catch different requests using the get or post method and routes them to the specified folder or function. So we can create an entire application with Express. However, transferring data from server to client or vice versa is a bit complicated so we will not manage it with Express. Socket.io handles this problem very well. It makes use of web sockets to transfer the data between the server and the client. There are two kind of connections we can make using Socket.io.

1. Persistent connection
2. Non persistent connection

In a non persistent connection the server opens a new socket on each data transfer. This method consumes a lot of time so it is not worth to use it in our application because for us time is gold. So we have decided to create a persistent connection. A persistent connection means maintaining a web socket opened all the time. To achieve this we must use the Socket.io even in the client-side.

IV. DATA ANALYSIS

As we mentioned earlier in this article, we will test Node.js against Php/ Apache. In this case we will get data from Yahoo! Finance which are:

1. Company Name
2. Ask (real time)
3. Bid (real time)
4. Change

We have used Yahoo! Finance as a provider for this data because it has a faster response time than other providers like Trade King. For example, TradeKing make use of a library (oauth) which authenticates the request before the server respose. This procedure increase the server response time but we don't want this to happen, that's why we use Yahoo! Finance as a provider.

We need to make a request to the Yahoo! sever in order to retrieve this data. The server responds with a .csv file on which we will work out later.

The Node.js application

Initially we must create the http server to serve this data to the users. The code to make this possible is like below.

```
var express = require('express'),
path = require('path'),
http = require('http'),
io = require('socket.io'),
var app = express();

app.configure(function () {
app.set('port', process.env.PORT || 3000);
app.use(express.logger('dev'));
app.use(express.bodyParser())
```

*Corresponding Author: Krisel Tola

Computer Science Department, "Aleksander Moisiu" University of Durres Albania

```
app.use(express.static(path.join(__dirname, 'public')));
});
var server = http.createServer(app);

io = io.listen(server);

server.listen(app.get('port'), function () {
console.log("Express server listening on port " + app.get('port'));
});
```

As we can see from the code above, we use express from the beginning. This framework helps us on bringing up our server. After that we need to configure the port on which the server will listen. Again, with the help of Express, we specify the directory on which the files that will be served stands and also we connect Socket.io with the newly created server. After setting up the server we need to make a request to the Yahoo! Finance.

```
var request_options = {
host: 'download.finance.yahoo.com',
port: 80,
/* To understand the Yahoo Finance query scheme, see
* http://www.gummy-stuff.org/Yahoo-data.htm
*/
path: '/d/quotes.csv?s=AAPL+GOOG+GLD+MSFT+CAS+AOS+ACM+AHC+AIR+CSCO&f=sb2b3c1'
};
```

We initially specify the host that will serve to our requests. The symbols joined with + are the symbols that represents the companies for which we want to get those data. The characters after "f" are the data options we will retrieve. After getting those data we use a function to convert them from .csv format into an array. We now need to transport this data from the server to the client so it is now the time that Socket.io framework comes in the help.

Socket.io on the server side

```
io.sockets.on('connection', function(socket) {
socket.on('stock_update_request', function(data) {
socket.emit('update_stock', {stocks: data_array});
});
socket.on('stock_init_request', function(data) {
socket.emit('get_stock', {stocks: data_array});
});
});
```

Socket.io on the server side

```
socket.on('update_stock', function(data) {
saved_data = data.stocks;
for(var j = 0; j < 10; j++){
$(".cmpask"+j).html(saved_data[j][1]);
$(".cmpbid"+j).html(saved_data[j][2]);
$(".cmpchg"+j).html(saved_data[j][3]);
}
});

socket.on('get_stock_init', function(data){
saved_data = data.stocks;
console.log(saved_data);
for(var i = 0; i < 10; i++)
{
var render = '<div style="float:left; margin:10px 10px 0 0"><p><span>Company Name :</span><span
class="cmpname'+i+'">'+ saved_data[i][0] +'<span></p> <p><span>Ask :</span><span
```

*Corresponding Author: Krisel Tola

Computer Science Department, "Aleksander Moisiu" University of Durres Albania

```
class="cmpask'+i+'">'+ saved_data[i][1] +'</span></p> <p><span>Bid : </span><span
class="cmpbid'+i+'">'+ saved_data[i][2] +'</span></p> <p><span>Change : </span><span
class="cmpchg'+i+'">'+ saved_data[i][3] +'</span></p> </div>';
$("#stocks").append(render);
}
});
```

In the code above, we have established a transmission line from client to server and vice versa, using socket events. This connection is also called a persistent connection because both sides send and receives data. To update the data every second we use the code below.

```
setInterval(function()
{
socket.emit('stock_update_request');
}, 1000);
socket.emit('get_stock_init');
```

V. THE PHP APPLICATION

As Php rely on Apache we do not need to create a server here. Coding with Php Language is of course much easier, because this language has a wide range of support and a lot of useful, ready to use, functions. We use the code below to make a request to the server and to convert this data from .csv format into an array.

```
$url="http://download.finance.yahoo.com/d/quotes.csv?s=AAPL+GOOG+GLD+MSFT+CAS+AOS+ACM+A
HC+AIR+KO&f=nb2b3c1";
$csv = file_get_contents($url);

$retrieve = array_map (
function ($_) {return explode (',', $_);},
explode ("\n", $csv)
);
```

Getting data from the array is not a rocket science. We can also integrate the html language into the Php script so we don't need to search for a folder with the files to serve here like we have done in the Node.js application. The code below explains that very well.

```
$data_array = update_and_retrieve::update_data();

$show = "";
for($i = 0;$i<10;$i ++){
$show .= '<div style="float:left; margin:10px 10px 0 0"><p><span>Company Name :</span><span
class="cmpname'.$i.'">'. $data_array[$i][0] .'<span></p> <p><span>Ask :</span><span class="cmpask'.$i.'">'.
$data_array[$i][1] .'</span></p> <p><span>Bid : </span><span class="cmpbid'.$i.'">'.
$data_array[$i][2] .'</span></p> <p><span>Change : </span><span class="cmpchg'.$i.'">'.
$data_array[$i][3] .'</span></p> </div>';
}
echo $show;
```

To update the data we have used a crone job.

As you can see we need less code to create the same application but the length of code does not indicate that this script is faster. Now it is time to test the apps. We have used Load Impact to do this. Both apps are tested against 50 simulated browser users for ten minutes. In Fig. 1 and Fig. 3 the green line shows the number of active users making a request to the servers while the blue line indicates the time that take the server to respond to the active users. Fig. 2 and Fig. 4 shows the average server response time. They also indicates the load time distribution between the scripts used in the applications.

*Corresponding Author: Krisel Tola

Computer Science Department, "Aleksander Moisiu" University of Durres Albania

Node application result

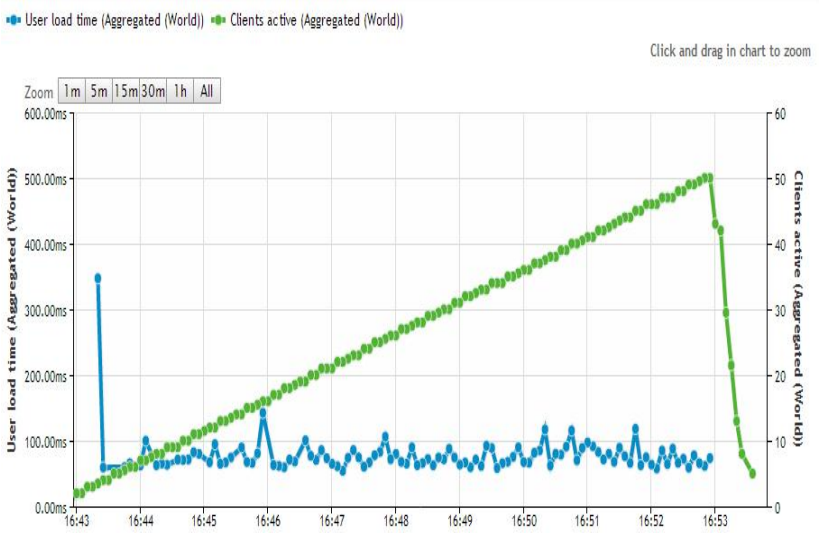

Fig. 1

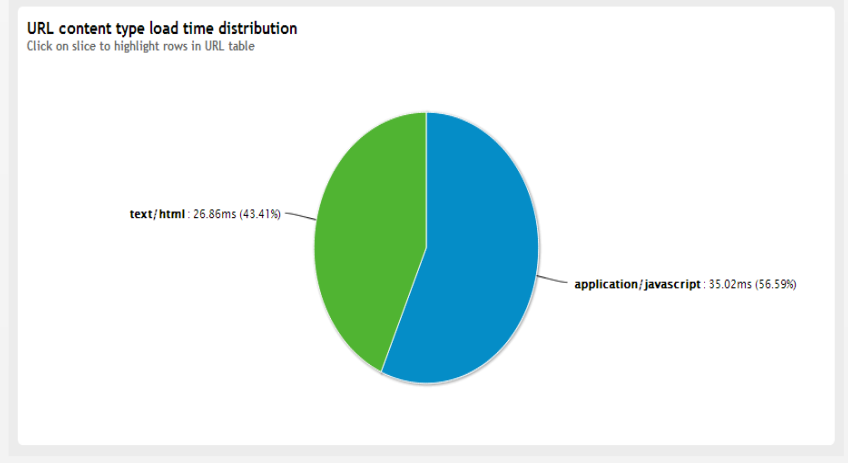

Fig. 2

Php application result

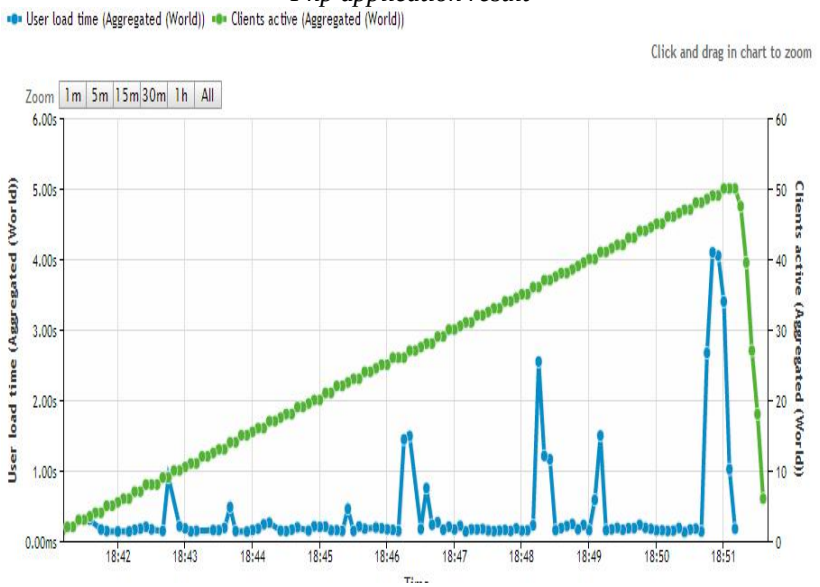

Fig. 3

*Corresponding Author: Krisel Tola
Computer Science Department, "Aleksander Moisiu" University of Durres Albania

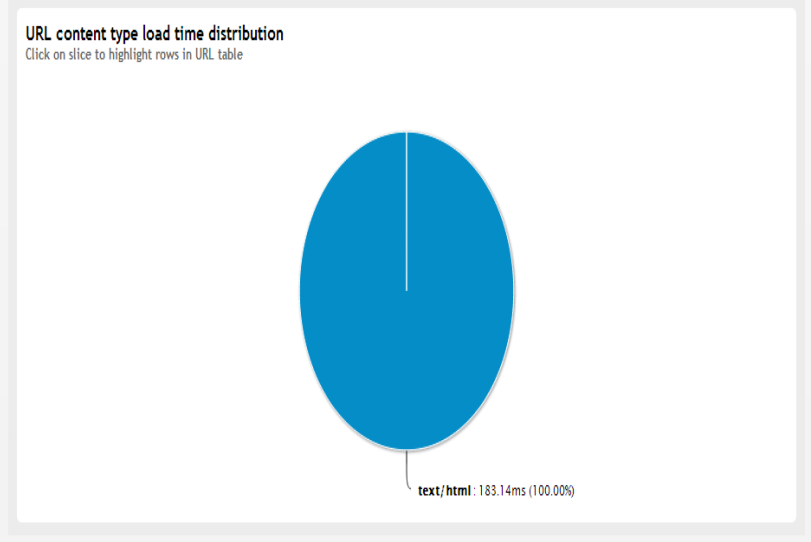

Fig.4

VI. CONCLUSIONS

The test results indicate us which of the application is fastest. As we can see from the images above, the Node.js application has an execution time of approximately 62 ms while the execution time of the Php/ Apache application takes about 183 ms to execute. Also the execution time of the Node.js appears more stable throughout the test. Based on the above results we conclude that the Node.js application is twice faster than the application created using Php/ Apache. The security that offers Php/ Apache is not a must for our application because we are not containing any confidential information of the users here and also we do not offer to our users any interactivity like transactions etc. The only goal of this application is showing up information to the users as fast as possible and we think the Node.js technology fulfill this quest. Based on these results we encourage the use of these technologies.

REFERENCES

[1]. http://nodejs.org/
[2]. http://en.wikipedia.org/wiki/Event-driven_programming
[3]. http://en.wikipedia.org/wiki/Asynchronous_I/O

*Corresponding Author: Krisel Tola

Computer Science Department, "Aleksander Moisiu" University of Durres Albania