Scalable and user friendly user interface for time-series analytics for OpenTSDB

Roberto Martín Muñoz
University of Stavanger
Department of Electrical and Computer Engineering
Stavanger, Norway
rm.munoz@sud.uis.no

Abstract—**The current web user interface of OpenTSDB is simplistic and not interactive. The aim of this paper will be to provide a more reliable and interactive architecture using visual analytics, NodeJS, websockets, and R. In this report we will introduce a beta version of the user interface with the main architecture implemented. This will be fully developed in a Spring 2014 master's thesis.**

Keywords-user interface; opentsdb; nodejs; websockets; R;

I. INTRODUCTION

OpenTSDB[1] is a scalable database built on top of HBase[2] and specifically designed for managing time series data. This database is used at the University of Stavanger (UiS) to store and access time series data such as daily weather and data from sensors around the building (Part of the project *Self learning Energy Efficient builDings and open Spaces* (SEEDS)). The great advantage of this database is its efficiency in managing a large amount of data points.

The OpenTSDB database has a web interface in which one can fetch data represented in plots. While it is adept at managing data, the interface is basic and does not allow the analyst to zoom in on the data points or navigate easily through them. Ultimately, this results in wasted time for the analyst, as they must change the query over and over again.

The requirements of this project will be the adoption of scalable access to analytic functions, that will be provided by a concurrent Master thesis written in R[3], and design, implement and evaluate a UI to enable easy access to analytic functions through a web interface.

To address the problem of the interface, we will provide a real dashboard based in the latest web standards that will allow an analyst to navigate through the data in a more interactive way and provide contextual information about the data. This will more efficiently detect patterns and gather more knowledge from the raw data. As a future task for the Master Thesis, we will implement an account manager with proper security for the system allowing the analyst to have their own personal account to store favorite plots, most used plots, favorites metrics, and custom alerts in datasets (like range restriction).

A NodeJS[4] server will be implemented to serve the page to the clients and fetch the points through an R script that we will use in the future to add error detection and time series prediction. That R code will fetch the data from OpenTSDB and then apply transformations if necessary.

The implementation will be modular and based on the three layers (analytics, data management, and visualization) that visual analytics must relay to meet analyst requirements[5]. In order to help the community behind OpenTSDB, the code will be published with an open source license.

To summarize,we will start with a theoretical overview of the technologies that will be used in the system, including a brief explanation OpenTSDB, NodeJS, websockets, Rserve, and several visualization frameworks. Then we will explain the architecture of the system as well as some metrics regarding its performance. We will finish with the current state of the system and the future implementations that will be done during the Master Thesis.

II. THEORETICAL FRAMEWORK

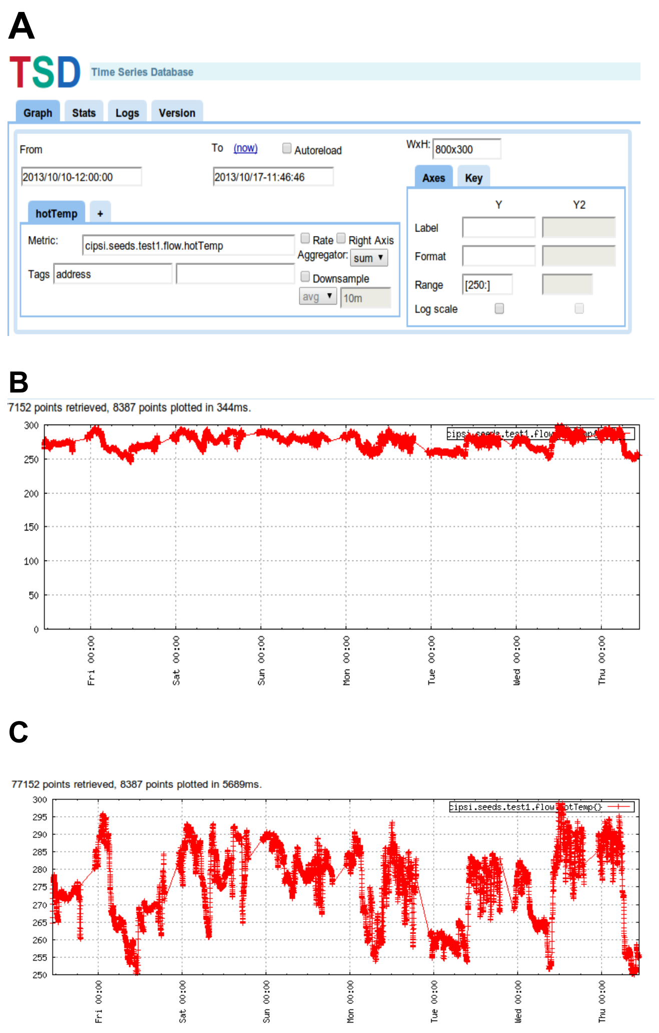
A. Current OpenTSDB web interface

The current interface consists of two main parts **(Figure 1)**: the form to enter the query **(Figure 1A)** and the resulting plot **(Figure 1B)**. When we access the interface the blank form shown in **(Figure 1A)** appears and is ready to accept parameters. Once the parameters are filled, a plot, such as the one seen in **(Figure 1B)**, appears. This plot presents a huge disadvantage to OpenTSDB users because it is only an image. If there were a mistake in entering the scale **(Figure 1C)** or size, the query would have to be executed again. With this system, the user is not able to zoom in or to get the correct scale automatically.

Moreover, the user is not able to save recurrent queries, interact in any way with the resulting plot, or apply transformations to the data directly in the interface.

A. Architecture of the solution

There are a variety of approaches that can be followed to address the user interface issues we have here. Visual analytics[6] is one of these approaches. Visual analytics is the science that studies the visual interactive interfaces that facilitate the reasoning of data to an analyst. In visual analytics we find the three-layers architecture[7] (**Figure 2**) that a dashboard system should have to fulfill an analyst's requirements in terms of analyzing data. The three layers (analytics, data management and visualization) share



From: 2013/10/10-12:00:00
To: (now) 2013/10/17-11:46:46
Autoreload
WxH: 800x300
Metric: cipsi.seeds.test1.flow.hotTemp
Tags: address
Rate
Right Axis
Aggregator: sum
Downsample: avg 10m
Label
Format
Range: [250:]
Log scale

7152 points retrieved, 8387 points plotted in 344ms.

77152 points retrieved, 8387 points plotted in 5689ms.

Figure 1 shows the current UI of the OpenTSDB system. A) shows the query window, B) the resulting plot, and C) the same plot in wrong scale (starting in 0)

information amongst them and integrate in two different workflows, reactive and interactive[8].

1) *Analytic layer*

The analytics layer, executes transformations on the raw data like statistical analysis, predictive behaviour or error detection and draws conclusions from it.

As one of the requirements is use analytics functions written in R,it is possible to implement the analytic layer by using R. R is a programming language focused in statistical analysis and data mining that is widely used by large companies like Google, Facebook or Microsoft[9].

2) *Data management layer*

The data management layer is responsible for the data lifecycle and procedures of the systems. To implement this layer we will use a web server coded in javascript, NodeJS. Node allows us to use javascript to create and manage web servers. It is a wrapper around the V8 JavaScript compiler of Google Chrome[10].

V8 is a highly optimized javascript engine that powers Google Chrome and it is the base of Node, adding some Node bindings like sockets and the node standard library that adds more features to V8 to make it a real server solution. The main features that make Node very attractive to deploy web-based applications are asynchronous I/O operations, memory and CPU efficiency, and a strong concurrency handler.

Node is not the first implementation of a server in javascript. However, this implementation has grown exponentially in the last year and has a demonstrated degree of success. The community surrounding Node is growing each day and this popularity is seen in all the modules that are being released into Node, from security modules to real time streaming features. Node has proven high reliability and performance managing web servers. A quick comparison with an Apache web server give us the conclusion that using node is faster and more scalable[11]. This technology is being used now by relevant companies such as Google in Google +, Microsoft in Windows Azure interface, or Yahoo in Yahoo Manhattan[12].

3) *Visualization layer*

The visualization layer is the last layer and is in charge of taking the amount of points and plot it in a optimal way for the analyst. In web environment, it will refer to the final HTML web page that the analyst will see and interact. In the web page we will use plotting libraries in javascript that could handle the data points and the interactions of the user.

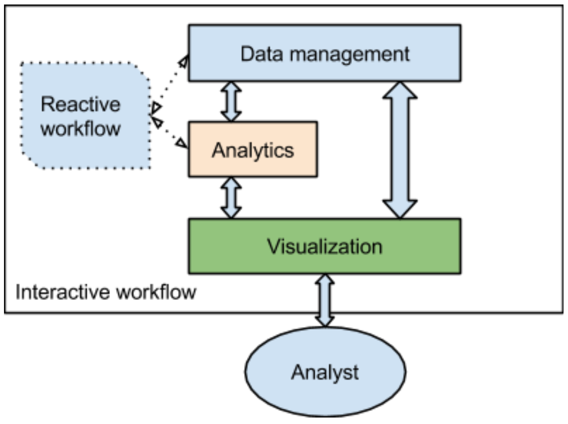


Figure 2: Three layers architecture of visual analytics applications with their workflows.

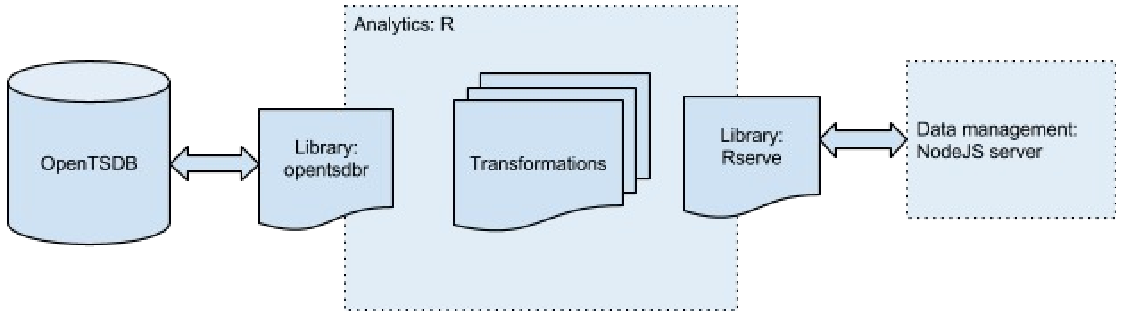


Figure 3: Diagram of the current implementation of the analytic layer with the libraries involved.

4) Workflows: Reactive and Interactive

The reactive workflow run operations ahead of time to prepare data analysis to be ready when the analyst request it. We can see this workflow in action in real time applications that need to perform analysis of the data when it arrives without interaction of an analyst. For example in processing images from a camera in real time, the analysis could be perform in the background while the system gets more images.

The interactive workflow run operations when the analyst request the data, it could be as simple as show the raw data, or as complex as the analyst request. This workflow requires interaction of the analyst.

B. Other architectures

Another interesting approach to the problem was using a framework called Shiny[13]. This is a complete web solution for R where with R code you can output HTML and javascript with plots. It doesn't need to use Rserve as R server or NodeJS middle server.

III. METHOD

Because of its demonstrated power in optimizing UI issues, we decided to use visual analytics to address our problem of the current user interface.

A. Implementations of the three layers

To implement the analytics layer, we use R as to apply transformations on the data. In the current implementation we only use it to fetch data from the database (OpenTSDB) without applying any algorithms, but in further implementations we will use predictions and error detection algorithms as well as statistical analysis.

There are two main challenges in the analytic layer, how we connect to OpenTSDB and how we serve that data to the data management layer.

To connect to OpenTSDB we have two libraries in R, opentsdbr[14] and R2Time[15]. Opentsdbr is simple library that will allow us to only read values in OpenTSDB with R code. Uses HTTP for the request being very inefficient but simple to use. On the other hand we have R2Time a library developed by Bikash Agrawal, PhD student of the University of Stavanger. It is a more optimized and complex library that go directly to HBase to fetch the data. For the purpose of the building the first version of the solution, we will use opentsdbr.

On the other side of the analytic layer, we need to serve the data to data management, for that we will use the R library Rserve[16], a library that will create a TCP/IP server in R so the data management layer could make queries with R code. The data management layer will send request to the server and Rserve will translate that requests in R code, execute it and give back the result.

In **Figure 3** we can see the diagram of the current impementation. In order to communicate with Rserve in the analytics layers, we need an Rserve client built in javascript and made for Node. There are three libraries that meet our requirements: Rserve-client[17], Rserve-js[18], and Node-RIO[19]. Rserve-client meets the requirements, however it cannot retrieve plots directly from Rserve. Moreover, the last activity in the code was from five months ago. It would be unwise to trust an abandoned library. Another viable option, Rserve-js, is library that has more functionality that Rserve-js, yet does not integrate well with error handling and is not optimized for managing large amount of data. Finally, Node-RIO is a complete library and compensates for all of Rserve-js's issues. It can work with error handling and is optimized for speed in data transfer.

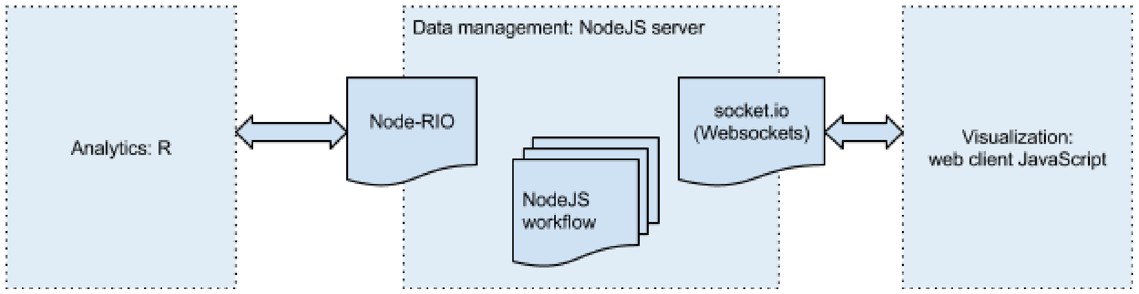


Figure 4: Diagram of the current implementation of the data management layer with the libraries involved.

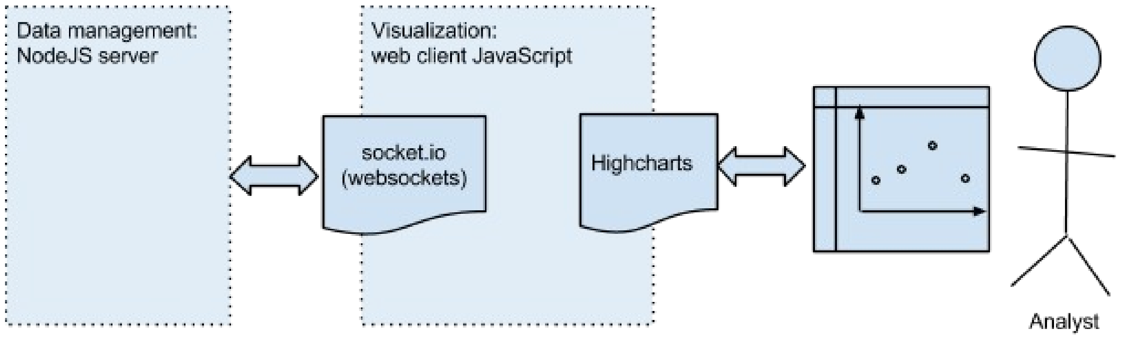


Figure 5: Diagram of the current implementation of the visualization layer with the libraries involved.

The three libraries operate in the same principle, fetching points from Rserve and converting it in a JSON object so it can be manipulated in javascript easily. For our solution, we have chosen Node-RIO because of its performance and error handling.

The last library that we are using in this layer is socket.io. It is a library for server and client (will be present also in the visualization layer) that implement websockets[20]. Websockets is a protocol that provides full-duplex communications over TCP connection, making real time communication between server and client fast and reliable. We will use this to send in an asynchronous way the data points retrieved from the analytics layer to the visualization layer in the client.

We chose websockets instead of using AJAX[21] mainly because of performance. Websockets is a new protocol designed for ensure fast transmissions between server and client. AJAX is a HTTP protocol and requires using the handshake protocol making the requests messages bigger than the response[22]. We need to transfer as much amount of data as possible, so the performance of AJAX protocol would be very bad.

In **Figure 4** we can observe that we have used two libraries in NodeJS, Node-RIO and socket.io[23]

In our implementation, the visualization layer (**Figure 5**) will be the web page that the data management layer, provides to the analyst. There will be a couple of

frameworks involved, one to received the data through web sockets and a framework to actual plot the data.

The socket.io library was explained before, here we will have the client part of the library to comunicate through websockes with with data management layer.

In the other side we have the plotting libraries, there are plenty of them and we evaluated some of them according of the requirements of the project:

Table 1: Comparison of visualization libraries

	Interactiveness	Mobile friendly	Popularity	Community	Time series specialization
gRaphaël [24]	Medium	--	Medium	Medium	Low
JavaScript InfoVis Toolkit [25]	Medium	Medium	Low	Low	Low
milkchart[26]	Low	Medium	Low	Low	Low
jQuery Visualize Plugin [27]	High	High	Low	Low	Low
moochart [28]	Low	Medium	Low	Low	Low
JS Charts [29]	Low	Medium	Low	Low	Low
Timeline [30]	Medium	High	Low	Medium	Medium
D3js [31]	High	High	High	High	High
Highcharts [32]	High	High	High	High	High

After taking a look to these library the only two that meet the requirements are D3js and Highcharts. For the purpose of the project, we will use Highcharts as visualization engine for proving the architecture but we won't discard using D3js in the Master Thesis. In **Figure 6** we can see an example of a zoomable time series plot using Highcharts.

B. Implementation of workflows

In the current implementation of our visualisation system, we are not going to implement the reactive workflow because of its complexity but in the future, when adding functionality like error detection and notifications, the reactive workflow will run in the background monitoring the dataset and analysing it to detect errors and if so, notify the user.

The interactive workflow is completed integrated and implemented in our system. It is the interaction of an analyst with the system. When the analyst request the main page, that interaction activate the workflow in the server, to gather the data and send it back to the analyst.

In the point (1) of **Figure 7** the server send empty plots and the client ask through websockets to get the data points of a specific plot id, the server get the id and query to Rserve the data requested. Once the server has the data it is sent by the same websocket of the request. One of the advantage of using NodeJS and websockets is that is not blocking, we can put a progress bar per plot or another mechanism to minimize the impact of waiting for the results.

C. Other architectures

The two main disadvantages and the reasons why we discarded Shiny was that don't follow the three layers architecture,mixing analytics and data management, and the second disadvantage that is very simple. The plots are not interactive because they are restricted to R plots, and it is very close in terms of customization. It is not suitable for a high front end interface for OpenTSDB.

Table 2: Comparison of alternatives UIs

UI	Interactivitiy	Documentation	R compatibility	Customizable
StatusWolf	High	Medium-Low	None	Medium
Otus	Medium - Low	Medium	None	Low
Tsdash	Medium	Low	None	Low
OpenTSDB -dashboard	--	Deprecated	--	--

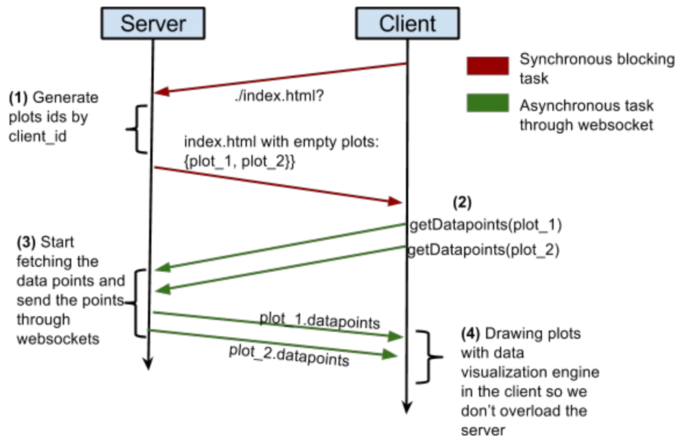

Server
Client
Synchronous blocking task
Asynchronous task through websocket
(1) Generate plots ids by client_id
./index.html?
index.html with empty plots: {plot_1, plot_2}}
(2)
getDatapoints(plot_1)
getDatapoints(plot_2)
(3) Start fetching the data points and send the points through websockets
plot_1.datapoints
plot_2.datapoints
(4) Drawing plots with data visualization engine in the client so we don't overload the server

Figure 7: Interactive workflow

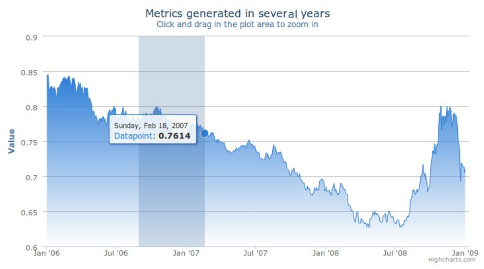


Figure 6: Example of plotting data points with more information

This is not the first attempt to build a better interface for OpenTSDB. There are another alternatives as UI for OpenTSDB[34]

D. *Driven test*

In order to ensure the quality of the system, after working of a complete first version of the system, we will organise a driven test in February with several analysts that are currently using the current web interface of OpenTSDB to collect information about their actual experience and explore the new dashboard and, if necessary, modify the design according to their feedback. This will be done along with personal interviews to gather as much data as possible.

IV. Results

In order to test the architecture we have done two sets measures with different amount of data points to see if it is possible to scale this architecture and add more interesting features.

The dataset used is one that we have in the OpenTSDB managed by the University of Stavanger, and this dataset will be the final database used along the interface.

We created four sets of scenarios (set-0, set-1, set-2 and set-3) which the difference between them is the amount of data points (different time frames). The first (set-0) from 2013-08-04 12:00:00 to 2013-08-07 14:00:00, the second (set-1) one from 2013-08-04 12:00:00 to 2013-08-11 14:00:00, the third one (set-2) from 2013-08-04 12:00:00 to 2013-08-19 14:00:00, and the last one (set-3) 2013-08-04 12:00:00 to 2013-08-26 14:00:00. The total number of data points in the first case is 1357 (three days, set-0), the second case 3047 (seven days, set-1) , in the third case 7163 (15 days, set-2), and in the last case 11204 (22 days, set-3) representing the temperature measure by a sensor.

We will do 4 tests each time, one after the other one, of the 4 main points represented in **Figure 7**

1) Time from getting request from client to send the page with empty plots.

Point (1) of **Figure 7**. At this stage, the server only creates the HTML page and send it to the client. This is the only petition that the client has to wait for the HTML.

This test is independent of the number of points, because the empty HTML is the same for both. As we can see the server notice that the second time that the page is requested, is the same page as the first time so use the page in cache, being 3 times faster than in the first case

Table 2: Milliseconds used to send the empty HTML to the client

Sets	Test 1	Test 2	Test 3	Test 4
set-0	17 ms	7 ms	5 ms	4 ms
set-1	16 ms	8 ms	6 ms	5 ms
set-2	18 ms	5 ms	5 ms	3 ms
set-3	16 ms	7 ms	5 ms	4 ms

Table 3: Time to process the empty plots

Sets	Test 1	Test 2	Test 3	Test 4
set-0	1 ms	1 ms	0 ms	0 ms
set-1	1 ms	1 ms	1 ms	1 ms
set-2	1 ms	0 ms	0 ms	1 ms
set-3	1 ms	1 ms	0 ms	1 ms

Table 5: Time required from the library opentsdbr to obtain the data points form OpenTSDB

Sets	Test 1	Test 2	Test 3	Test 4	Average
set-0	709 ms	714 ms	774 ms	722 ms	729.75 ms
set-1	2085 ms	2002 ms	2032 ms	2060 ms	2044.75 ms
set-2	5081 ms	5052 ms	5626 ms	5607 ms	5341.5 ms
set-3	8331 ms	8423 ms	8571 ms	8272 ms	8399.25 ms

Table 4: Time to obtain the data points form OpenTSDB

Sets	Test 1	Test 2	Test 3	Test 4	Average
set-0	1992 ms	1974 ms	2010 ms	2041 ms	2004.25 ms
set-1	3446 ms	3366 ms	3344 ms	3402 ms	3389.5 ms
set-2	6414 ms	6424 ms	7053 ms	6902 ms	6698.25 ms
set-3	9683 ms	9728 ms	9930 ms	9585 ms	9731.5 ms

2) Time to process the plots in the client and request the data points through websockets to the server

Point (2) of Figure 7. This is a non blocking call and basically the client go through all the empty plots and request the data points based in the id of the empty plot.

As seen above, this test is independent of the amount of points. Is a simple process and does not require much time.

3) Time of the server Node doing the request to the Rserve and getting back the points

Point (3) of **Figure 7**.
This is the total time used by the Node server to obtain all the points and get them ready to send to the client (passing all the analytics layer):

This times are high, so we decided to see who is the responsible of the delay. Then we decided to take a look at the time needed in R to do the petition to OpenTSDB with the library opentsdbr.

We can see in the **Figure 8** that most of the time is used by the library opentsdbr to query the database.

With these results we conclude that the library opentsdbr is the bottleneck of the whole system and in future developments we will test the other library R2Time to see its performance under the same circumstances

4) Time from getting the points in the client to finish plotting
Point (4) of **Figure 7**.
This is the time that the visualization library Highcharts takes to process 7163 data points and plot them.

These results are reasonable thinking the amount of data points managed but in future developments we will compare this results to the results of that the other visualization library, D3js under the same circumstances.

Table 6: Time required from the library Highcharts to process and plot all the data points.

Sets	Test 1	Test 2	Test 3	Test 4
set-0	110 ms	115 ms	120 ms	113 ms
set-1	124 ms	141 ms	129 ms	133 ms
set-2	227 ms	192 ms	218 ms	208 ms
set-3	239 ms	225 ms	240 ms	237 ms

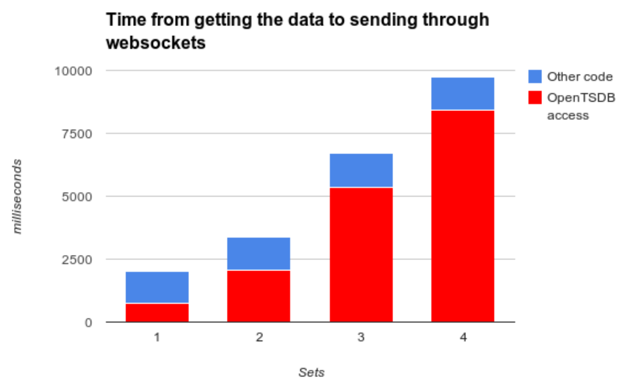


Figure 8: Comparison between total time and time of the library opentsdbr between all the sets

V. Conclusion

This paper set out to find the correct and optimal architecture instead of the current one of OpenTSDB. Through the three layers architecture, we have found that it is possible to create a better and more interactive user interface. These findings are important contributions to help analyst to perform their tasks efficiently.

In terms of future projects, we will particularly focus on complete the user interface adding new functionalities to create a system that fulfill an analyst needs.

VI. Acknowledgements

Foremost, I would like to express my gratitude to my advisor, PhD Associate Professor Tomasz Wiktor.

My sincere thanks also goes to Tatiana Popovitchenko for editing.

VII. References

[1] "OpenTSDB - A Distributed, Scalable Monitoring System." 2010. 11 Dec. 2013 <http://opentsdb.net/>

[2] "HBase - Apache HBase™ Home." 2010. 11 Dec. 2013 <http://hbase.apache.org/>

[3] "The R Project for Statistical Computing." 12 Dec. 2013 <http://www.r-project.org/>

[4] Tilkov, Stefan, and Steve Vinoski. "Node. js: Using JavaScript to build high-performance network programs." Internet Computing, IEEE 14.6 (2010): 80-83.

[5] Thomas, James J., and Kristin A. Cook, eds. "Illuminating the path: The research and development agenda for visual analytics." (2005).

[6] Thomas, Jim, and Pak Chung Wong. "Visual analytics." IEEE Computer Graphics and Applications 24.5 (2004): 0020-21.

[7] Thomas, James J, and Kristin A Cook. Illuminating the path: The research and development agenda for visual analytics. James J Thomas & Kristin A Cook. IEEE Computer Society Press, 2005.

[8] Fekete, Jean-Daniel. "Visual Analytics Infrastructures: From Data Management to Exploration." Computer 46.7 (2013): 22-29.

[9] "Companies Using R | Revolution Analytics." 2013. 17 Dec. 2013 <http://www.revolutionanalytics.com/companies-using-r>

[10] "node.js." 2009. 14 Dec. 2013 <http://nodejs.org/>

[11] "Benchmarking Node.js - basic performance tests against Apache + ..." 2011. 14 Dec. 2013 <http://zgadzaj.com/benchmarking-nodejs-basic-performance-tests-against-apache-php>

[12] "Projects, Applications, and Companies Using Node · joyent ... - GitHub." 2011. 14 Dec. 2013 <https://github.com/joyent/node/wiki/Projects,-Applications,-and-Companies-Using-Node>

[13] RStudio - Shiny." 2012. 16 Dec. 2013 <http://www.rstudio.com/shiny/>

[14] "holstius/opentsdbr · GitHub." 2013. 14 Dec. 2013 <https://github.com/holstius/opentsdbr>

[15] Agrawal, Bikash. "Analysis of large time-series data in OpenTSDB." (2013).

[16] Urbanek, Simon. "Rserve--A Fast Way to Provide R Functionality to Applications." PROC. OF THE 3RD INTERNATIONAL WORKSHOP ON DISTRIBUTED STATISTICAL COMPUTING (DSC 2003), ISSN 1609-395X, EDS.: KURT HORNIK, FRIEDRICH LEISCH & ACHIM ZEILEIS, 2003 (HTTP://ROSUDA. ORG/RSERVE 2003.

[17] "rserve-client - npm." 2012. 18 Dec. 2013 <https://npmjs.org/package/rserve-client>

[18] "cscheid/rserve-js · GitHub." 2013. 17 Dec. 2013 <https://github.com/cscheid/rserve-js>

[19] "albertosantini/node-rio · GitHub." 2011. 17 Dec. 2013 <https://github.com/albertosantini/node-rio>

[20] Lubbers, Peter, and Frank Greco. "Html5 web sockets: A quantum leap in scalability for the web." SOA World Magazine (2010).

[21] "Ajax (programming) - Wikipedia, the free encyclopedia." 2005. 16 Dec. 2013 <http://en.wikipedia.org/wiki/Ajax_(programming)>

[22] "HTML5 WebSocket vs Long Polling vs AJAX - Stack Overflow." 2013. 16 Dec. 2013 <http://stackoverflow.com/questions/10028770/html5-websocket-vs-long-polling-vs-ajax>

[23] "Socket.IO: the cross-browser WebSocket for realtime apps." 2010. 16 Dec. 2013 <http://socket.io/>

[24] "gRaphaël—Charting JavaScript Library." 2009. 16 Dec. 2013 <http://g.raphaeljs.com/>

[25] "JavaScript InfoVis Toolkit - Nicolas Garcia Belmonte." 2013. 16 Dec. 2013 <http://philogb.github.io/jit/>

[26] "MooTools Forge | MilkChart." 2009. 16 Dec. 2013 <http://mootools.net/forge/p/milkchart>

[27] "Update to jQuery Visualize: Accessible Charts with HTML5 from ..." 2010. 16 Dec. 2013 <http://filamentgroup.com/lab/update_to_jquery_visualize_accessible_charts_with_html5_from_designing_with/>

[28] "moochart - charts for mootools." 2008. 16 Dec. 2013 <http://moochart.coneri.se/>

[29] "JS Charts - Free JavaScript charts." 2007. 16 Dec. 2013 <http://www.jscharts.com/>

[30] "SIMILE Widgets | Timeline." 2009. 16 Dec. 2013 <http://www.simile-widgets.org/timeline/>

[31] "D3.js - Data-Driven Documents." 2010. 16 Dec. 2013 <http://d3js.org/>

[32] "Highcharts - Interactive JavaScript charts for your webpage." 2007. 16 Dec. 2013 <http://www.highcharts.com/>

[33] "Alternative front ends · OpenTSDB/opentsdb Wiki · GitHub." 2013. 18 Dec. 2013 <https://github.com/OpenTSDB/opentsdb/wiki/Alternative-front-ends>