July 08, 2019

A WEBSOCKET INDEPENDENT REAL TIME VISUALIZATION OF HIGH FREQUENCY TELEMETRY DATA

HP INC

A WebSocket independent Real Time Visualization of High frequency telemetry data

With the magnitude of telemetry data constantly increasing, effective data analytics requires efficient, scalable and multiple representation of data and its dependencies. Telemetry data from printers can be used in several different ways to enable meaningful analytics. We have focused on two use cases

(i) Real time analysis:

- Display last N seconds for real-time monitoring and analysis
- Displaying full data set for the current session. i.e. if a session starts at t_0 and user connects at t_1 then they can see the full dataset starting from t_0 to $t_{current}$ which is integral for analysis. With WebSockets on the other hand, a user can see data streamed from t_1 to $t_{current}$ but, will not be able to visualize data from t_0 to t_1.

(ii) Display the entire data set for post print analysis.

- Users can view the full data set of any past session for analysis. A persistent connection is not required to view this data as in case of WebSockets as the data is already stored in server,

Graphical representations are critical abstractions for effective analysis of telemetry data. The uniqueness of our methodology lies in enabling both the above-mentioned use cases to be consumed by any graphing client simultaneously, which is not feasible with WebSockets. In addition, it is mandatory to maintain a persistent connection between the client and the server to fetch data while using the web socket approach which increases overhead on the server. The Proposed solution completely decouples the client and the server thereby creating a highly versatile analytics solution.

Problems Solved

(i) Real time analysis:

- Displaying last N seconds for real-time monitoring and analysis
- Displaying full data set for current session.

(ii) Display the entire data set for post print analysis.

- Users can view full data set of any historic session for analysis.

Prior Solutions

Our initial approach was to use WebSockets since they are well known industry standards to stream high volume real-time data. However, though they cater to the need of real-time data visualization, they fail in certain situations such as follows:

1. When multiple data sockets are served for a single client, the data needs to be interleaved at the server and again distributed at the client. This causes a huge bottleneck in case of real-time bulk data served to multiple users.
2. WebSockets keep connections open for all the clients irrespective of the presence or absence of activity. This increases enormous load on server.

3. WebSockets can stream live data better but saving the same data for future reference causes an overhead.
4. Re-connection handling mechanism is required in case of WebSockets which is not required in the proposed solutions.
5. Extracting a sub-set of the whole data using programming languages leads to memory insufficiency due to the large volume of data and limited server space. This could lead to data loss in a high frequency streaming environment.

Proposed Solution

Our objective is to create a highly efficient data collection and analytics system that can be used, by a multitude of engineers simultaneously, to monitor and analyze different parameters real time and for analysis of historical data.

The proposed solution is to have a server collect the data from all the printers and process them for any web client to consume the data and provide a graphical representation at any desired refresh rate.

The proposed solution helps reduce the memory consumption by 80% and relies more on disk IO to generate multiple csv files which can be consumed by any third-party graphing library such as “Zingcharts”, “HighCharts”, “ChartJS” and others. It additionally completely decouples the data collection and data visualization which creates a highly efficient and versatile analytics solution. The telemetry data can arrive at any frequency and the charting libraries can auto refresh at their desired refresh rate.

Architecture Diagram

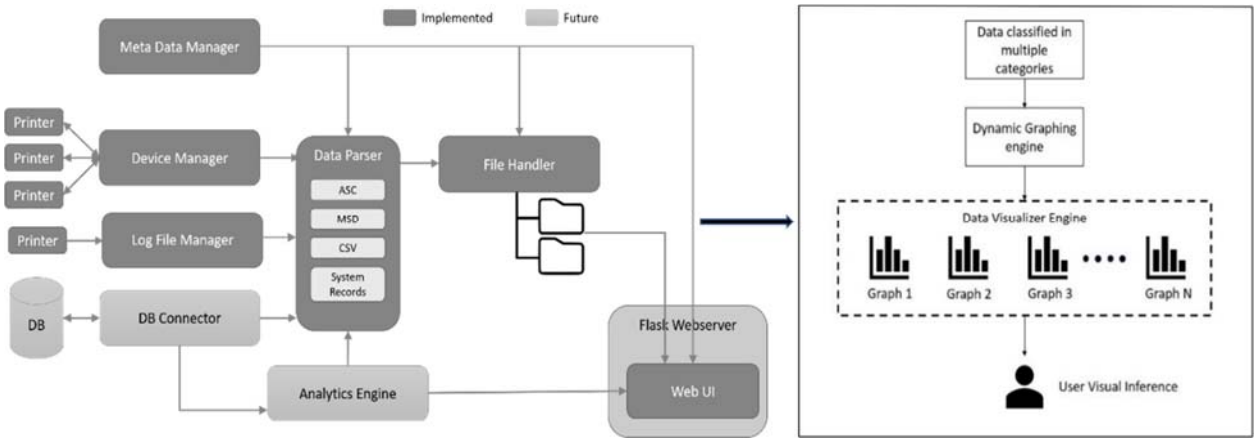

Figure 1

As explained in Figure 1, The data is received as interleaved stream of UDP packets from multiple devices. The device manager will manage all the devices connected to system. The Data parser is a collection of parsing logic and is highly specific. Each device can have its own set of parsers. The main functionality of the parser is to classify the data in different buckets. Any modeling or calculations can be applied at this layer. Once the parsing is done the data is sent to file handler which moves the data from the memory to csv files on specified checkpoints. The checkpoint can be a simple timer, example: write after every second or it can be a calculated field, example: write to a file when there are 3000 data points. This keeps the memory in check and data is processed and moved to its respective csv files as

and when it is received. The CSV files are structured in such a way that the first column represents the x-axis and the subsequent columns represent the data.

Any graphing library can load this file and plot the desired graph without extra processing. This approach allows us to process a single data point and assign it to multiple use-cases. For example, a data point will be part of a real-time graphing interval and when it moves past the interval the point is no longer a part of real-time visualization but is retained for historical analysis in another file. Visualizing real-time data for the last N seconds requires generation of truncated csv files to reflect only the last N seconds or last N number of points, which can be challenging.

The approach of using a programming language to achieve this is ineffective because of insufficient memory usage. Writing data to file and running a shell script at regular intervals solved this problem. A worker thread can initiate a shell script at each second. The shell script will truncate the files using Linux commands such as tail grep without causing any lag in the UI, creating a perfect moving data point stream. Since the commands used in shell scripts run directly on the kernel, they are significantly more efficient than any programming language. This is explained in Figure 2 below.

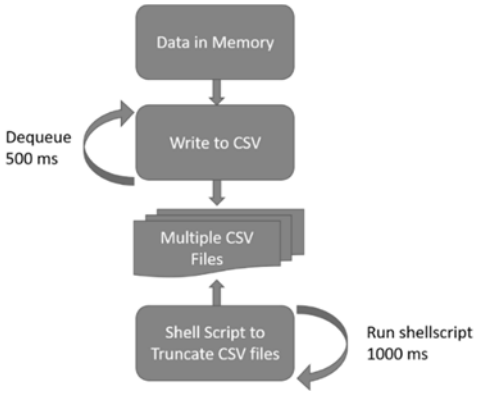

Figure 2

Advantages

The proposed approach involving the elimination of web sockets and allowing the client to directly fetch data from the server has multiple advantages in certain situations as follows:
1. Reduced network overloads caused due to open WebSockets by huge number of clients connected.

2. Streaming live data and saving the same for future reference simultaneously is seamless in the proposed solution (which was causing overhead in WebSockets)

3. Displaying full data set for current session. i.e. Even if a session starts at t_0 and user connects at t_1, they can see full dataset starting from t_0 to current time which is integral for analysis and was not possible using WebSockets.

4. Not limiting number of client connection as in case of WebSockets.

Disclosed by Damini Soni, Govindan Kumaresh, Surni Kumar and Sukanya Rengaswamy, HP Inc.