Time Series Updates for Traveler

List of my Pull Requests:

https://github.com/hdc-arizona/traveler-integrated/pull/59

https://github.com/hdc-arizona/traveler-integrated/pull/62

https://github.com/hdc-arizona/traveler-integrated/pull/67

Summary of last 4 weeks (July 1 – July 30):

In this google summer project, my goal is to make the Traveler interface more interactive and responsive by reducing visual lagging as well as providing more control (mouse interaction, file selector interface, etc.) over the frontend UI.

In the beginning of this period, I completed writing code to handle synchronous function calls which generated consecutive same events in the same hardware locality. This problem is handled by storing a stack of previous events and inserting new logical leave event in the place of consecutive enter events and vice-versa. This enables Traveler to read, parse, and visualize OTF2 logs with synchronous function calls. I also fixed a legacy bug in our system, which is, previously, we considered that the PAPI metric values would be same for the events of a single interval. More specifically, we stored metric values in a single JSON object for an interval. But the metric values could be different for an interval's enter and leave event. To resolve this issue, I used two separate JSON objects to store the metric values related to a single interval. This also helped to handle and manipulate the metric data more precisely since previously it was just losing metric events for leave events. On the frontend of the line charts, I have added dots to help users identify each value more precisely.

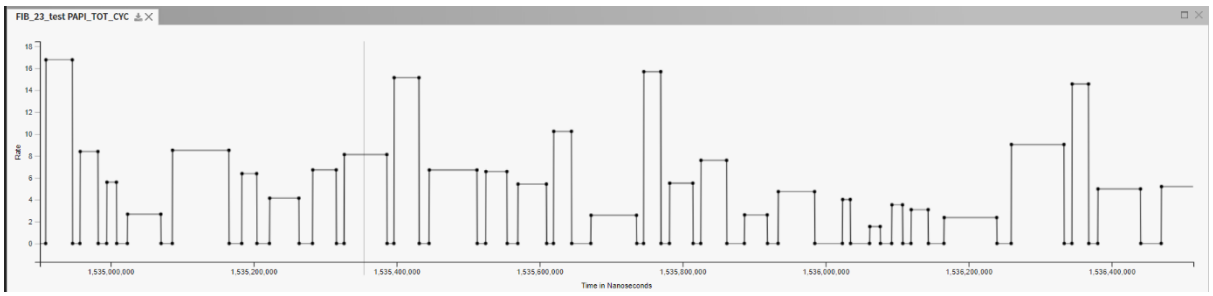

Figure: PAPI Metric (Total CPU Cycle)

To filter out different intervals, I have designed a new chart to contain a histogram of primitive durations. The new Histogram view presents the count of intervals of a specific duration on a logarithmic scale. This new view is acting as an accessory to filter out and highlight data in the Gantt View. In the backend, I have reused the summed area table data structured to store and process relevant information for the interval-durations per primitive. In the frontend, the user interface has been designed using HTML5 canvas. A brush has been provided to let users navigate on different time spans. Users could easily find out the time duration which is taking higher to complete for a particular primitive. Upon selection with the brush over the Histogram, intervals with related primitive is being highlighted with the same color in the Gantt View.

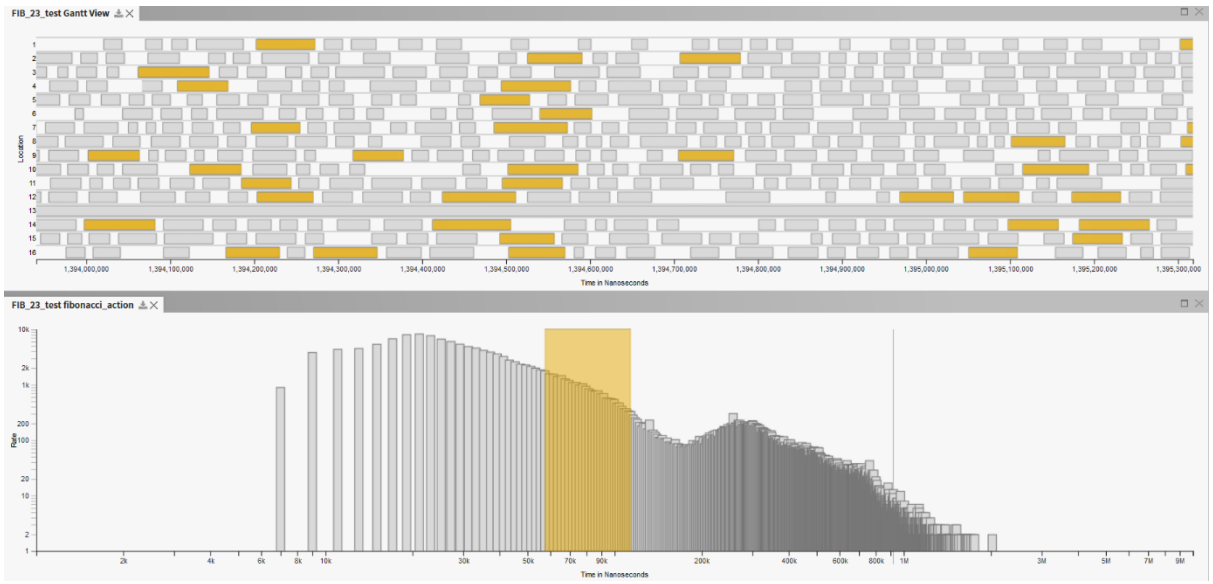

Figure: Primitive Histogram View

I have also added a hamburger menu on the left sidebar to browser and select a primitive from a list. Upon selection of a primitive, respective Histogram view will appear on the interface.

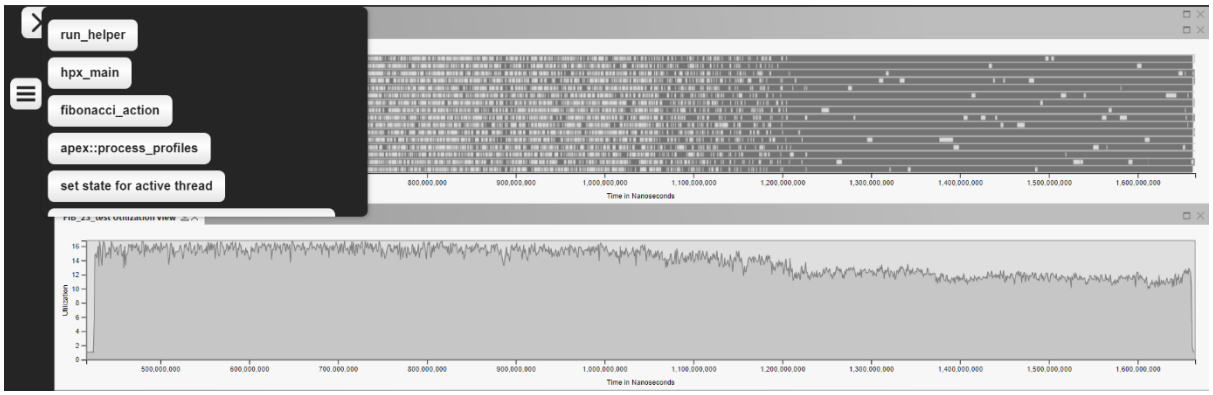

Figure: Hamburger menu containing Primitive names

List of completed new features in the second evaluation:

- Metric values for enter-leave events of an interval stored separately.
- New logical event added in place of each nested function calls.
- Summed area table technique reused (designed in the previous evaluation period) to fetch and organize data in the backend for the primitive histogram.
- New Primitive Histogram view created using HTML5 Canvas.
- Brush functionality implemented on the Histogram view.
- Related intervals highlighted in the Gantt View when brush modified.
- Newly included APIs:
    - *getPrimitiveList* (fetch primitive list for the hamburger menu)
    - getIntervalDuration (histogram data of each primitive duration)
    - getIntervalList (filtered data for the Gantt view)
- New locking mechanism implemented to synchronize interactive data highlighting in the Gantt view.