title: “Report - TERASCOPE_CSC8634” author: “Parth Umeshkumar Shah” date: “17/01/2022” output: pdf_document

Introduction

Terapixel images, this images allow viewers to browse big data across multiple scales in an interactive and intuitive manner, allowing stakeholders to interact with them. By computing realistic terapixel visualizations, Newcastle Urban Observatory captured a huge amount of environmental data about Newcastle Upon Tyne

http://terapixel.wasabi.1024.s3.eu-central-1.wasabisys.com/vtour/index.html

A major challenge :

A major challenge here was to find a way to deliver the supercomputer-scale resources required to compute the terapixels above. An example of a solution is to use a scalable architecture for cloud-based visualization, which can be deployed and paid for as needed. This work aims to achieve the following three objectives:

Develop a public cloud-based supercomputer architecture for scalable visualization; Visualization of terapixel 3D cities with daily updates; To conduct a rigorous evaluation of cloud supercomputing for visualization applications that are computationally intensive.

Based on a demonstration conducted, it was possible to produce high quality terapixel visualizations Following a path tracing renderer in under a day using public IaaS cloud GPU nodes. After being generated, the terapixel image supported interactive browsing across a range of sensing scales, from the city as a whole to a single desk within a room, and was compatible with a variety of thin client devices.

The performance of a supercomputer is further evaluated by storing data associated with terapixel generation.

A report based on analysis of the following information obtained from super cloud computer:

1. A “gpu dataset” contains timestamps when the gpu information is recorded, a hostname, a gpu serial number, and a gpu UUID representing the gpu node from which the specific observation was recorded, and parameters recording the gpu’s power, temperature, utilization, and memory usage.
2. “task-x-y dataset” consists of the task IDs of Azure batch tasks, the location of the terapixel rendered by the task and the Azure batch job ID.
3. “application checkpoints dataset” which consists the timestamps of taskId at specific checkpoints when an event starts or stops along with gpu node where the taskId was rendered.

Quick insights into datasets in detail :

In “application.checkpoints”, there are 660400 observations about 6 variables - timestamp, eventType, jobId, taskId, hostname,eventName

In “gpu” dataset, there are 1543681 observations about 8 variables - timestamp,hostname,gpuSerial, gpuUtilPerc, gpuMemUtilPerc, gpuUUID,powerDrawnWatt,gpuTemp

In “application.checkpoints”, there are 660400 observations about 6 variables - timestamp,hostname, jobId, taskId, eventName, eventType

In “task-x-y” dataset, there are 65793 observations about 5 variables - taskId,JobId,x,y,level

- Azure batch has 65793 taskId's, with 65536 tasks at level 12 terapixel, 256 tasks at level 8 and 1 at level 4. As there are only 1 task at level 4 here we have ignored values associated to it for analysis.
- In level 12, (x,y) coordinates are the positions of the tera pixel rendered with a unique taskId, at level 12, x and y range from 0 to 255.
- A JobId identifies what level tera pixel is being rendered at.
- GPU variables (power, temperature, utilization, and memory usage) are updated every 2 second - based on time difference of timestamps
- there are different 1024 gpu nodes encoded by the variables hostname, gpuSerial and gpuUUID
- The amount of time that is taken by gpu node to start rendering new task after completion of previous task is approximately 3 seconds.
- of 1543681 observations in gpu dataset there are 9 duplicates
- The range of gpu variables are as below

```
##  powerDrawWatt       gpuTempC      gpuUtilPerc     gpuMemUtilPerc
##  Min.   : 22.55   Min.   :26.00   Min.   :  0.00   Min.   : 0.00
##  1st Qu.: 44.99   1st Qu.:38.00   1st Qu.:  0.00   1st Qu.: 0.00
##  Median : 96.59   Median :40.00   Median : 89.00   Median :43.00
##  Mean   : 89.20   Mean   :40.08   Mean   : 63.06   Mean   :33.41
##  3rd Qu.:121.34   3rd Qu.:42.00   3rd Qu.: 92.00   3rd Qu.:51.00
##  Max.   :197.01   Max.   :55.00   Max.   :100.00   Max.   :83.00
```

- 5 events associated to every taskId and timestamps are logged at time when each of the events starts and stops rendering process.

Analyses based on initial insights and findings

- As there are 3 datasets, connecting them as a merged dataset to work based on taskId can be helpful to analyse data further
- It seems like the time stamp is crucial since it is a continuous variable that may aid in seeing the real-time progress of host computers as a whole.
- The minimum values of the gpuUtilPerc and gpuMemUtilPerc are 0 which suggests that there are chances of gpu staying idle.
- Process of joining task-x-y and application-checkpoints datset can be done by taskId which can opens option to analyze coordinates of rendered terapixel and runtimes associated with them.
- Also joining of data with gpu can be done through variables timestamp and hostname as they are the only common ones present. This leads resulting in many NA values which should be omitted further.
- If all the datasets are merged into a single data file, gpu performance from gpu variables can be linked with taskId, runtime, hostname and location of rendered terapixel
- Different required maps can be generated using x,y values with any continuous variable

Process of wrangling and preprocessing data

1. merging tasks_dataset and application checkpoints (rendering_dataset) by taskId as rendering-tasks_dataset.

2. With use of observations of gpu dataset,creating a unique dataset by combination of given 3 files as fully joining it with rendering-tasks_dataset by timestamp and hostname followed by ordering as full_dataset

3. full_dataset consists of all initial variables with NA values of variables in sets, observing it based on the dataset it is from either checkpoints_tasks or gpu datsets.

4. timestamp and taskId are key thread in the connecting datasets. full_dataset can be considered as log files with all the values recorded in time series. NA values of those taskId's before total render stops can be replaced with the last observed value as all the values of taskId between total render start and total render stop belongs to one taskId. Underlying assumption here is that each gpu host can only run one taskId at any instance. full_dataset is mutated by locf function (last observation carried forward) on taskId as priority and on x,y,level and jobId to ease the further analysis.

Note:full_dataset should be ordered by hostname and timestamp before using locf function.

5. In this step, the mutated data from the previous step is converted to a value on which arithmetic operations can be performed. timestamp is converted into a value which represents number of seconds since 00:00:00 hours:minutes:seconds of the day. It is important to check the date of timestamp. As the terapixel is rendered on the same day, considering the seconds passed from 00:00:00 will not be wrong, but it is very important to check before proceeding in this method.

example - 09:41:25 hr:mm:ss = 09x60x60+41x60+25 = 34885.

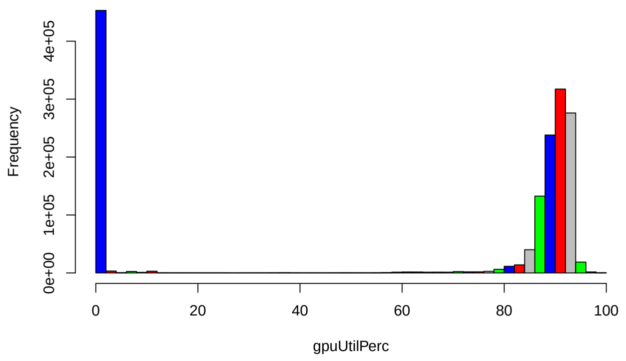
6. After the data obtained from the above step has been renamed & sorted, it is referred to as final_dataset.

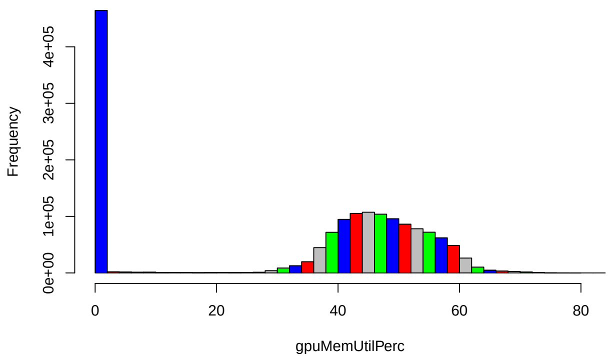
7. In order to calculate the runtime, subtracting the seconds_of_day of the START event from the STOP event when the connection between rendering, tasks & GPU datasets is generated. to perform this, pivot_wider operation is performed on a subset of final_dataset with variables taskId (as it is unique, it helps to join the outputs to final_dataset), eventName(5 application checkpoints), eventType(START,STOP) and seconds_of_day(important variable to substract). pivot_wider is performed with names from eventType and values from seconds_of_day. Once the pivotted data has been subtracted, the runtime or time_difference can be calculated,this data is then full joined with final_dataset by taskId and eventName as for every taskId there are only 2 eventNames by (start and stop) which can later be removed during specific analysis. the data generated from this is named as runtime_final_data.

8. The runtime_final_data is further manipulated by creating total power drawn, average gpu tempara-ture, average gpu utilization and average gpu memory utilization values specific to a taskId. taskId is preferred as for the complete analysis, taskId is a unique thread that can connect all the variables.

- The following assumptions are made when calculating the summarized GPU variables: if the GPU utilization per cent or the GPU memory utilization per cent are less than 15, it can be considered as idle performance'. the above assumption is taken as it is observed that when certain application checkpoints are in progress, there is a clear pattern that gpu performance variables seem to be running at considerable values, but after the rendering is completed or before the rendering starts, gpu usage is much less at 0 and very few are less than 15.

- The following is a histogram to compare the frequency of zero utilization. The values below 15 should be eliminated from the data since they will be counted as observations when finding average temperature or average utilization and the aggregation will pull the mean down very significantly.





- based on the assumption, gpu performance is calculated by mutating rows whose gpuUtilPerc or gpuMemUtilPerc are greater than 15 followed by summarizing the dataset grouped by taskId. while powerDrawnWatt is summarized by adding power drawn values within the group; gpuTempC, gpuUtilPerc and gpuMemUtilPerc are summarized by finding mean of values, the summarized data is named as final_gpu_performance.
- Post that final_gpu_performance is joined by taskId and named as analysis_dataset. Further exploratory data analysis is possible with this dataset, since it contains the information required to perform any analysis between all variables.

Graphical analysis and summaries

https://www.r-graph-gallery.com/79-levelplot-with-ggplot2.html

Q1- All Events Runtime analysis.

1. Runtime analysis based on events at level 8:

- each of the event are filtered at level 8 and heat map of runtimes specific to tasks are plotted over x,y coordiantes as the taskId have unique x, y coordinates.

- from the above graphs, saving configurations seems to be constant for all taskId's
- the heat map of rendering shows how good a time difference is present with respect to each taskID, moreover it shows how efficiently every pixel is rendered in super computers
- yes,for other 4 events apart from saving configuration runtime seems to have a wide contrast and nothing specific could be determined from the heat map.Hence, to understand runtimes further boxplots of runtimes are plotted as below:

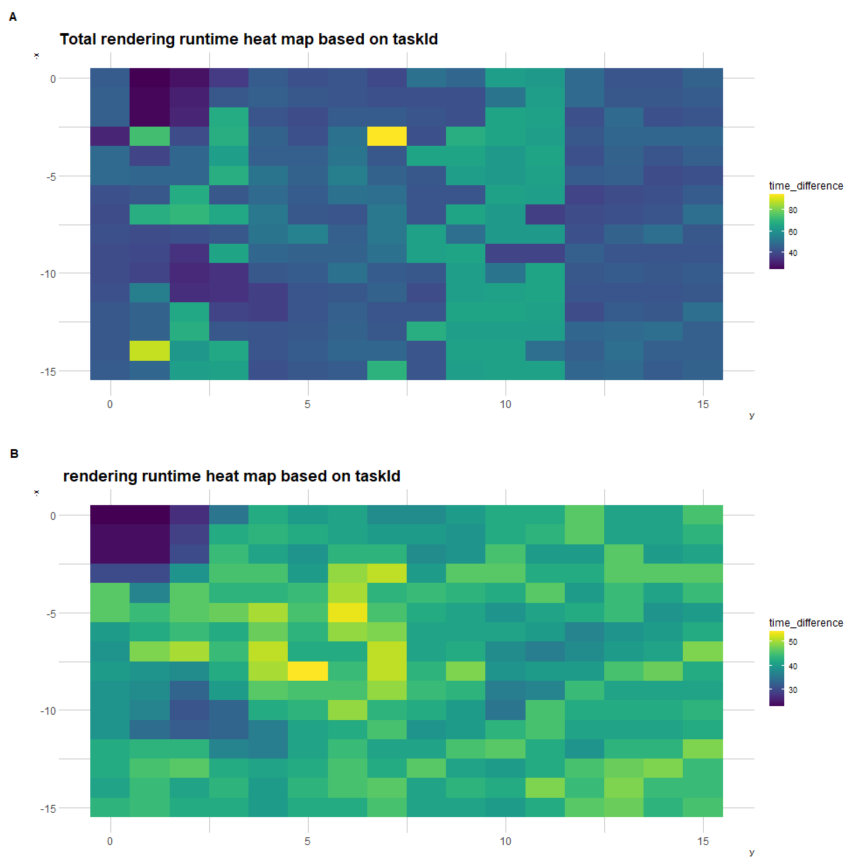


Figure 1: Analysis of desktop users based on total views of the modules

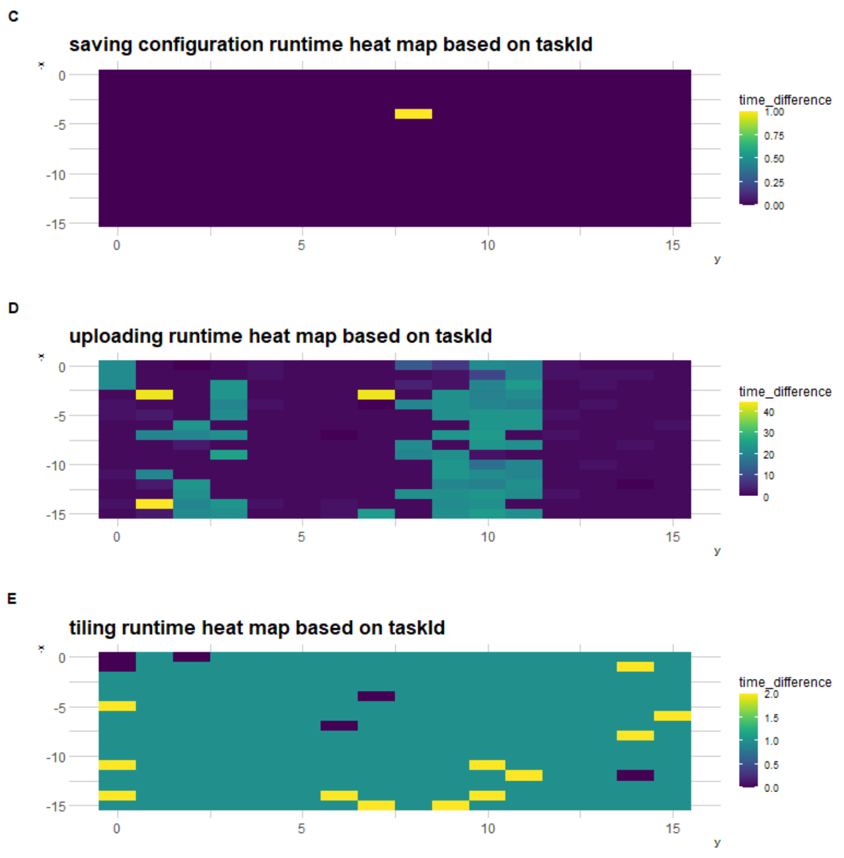
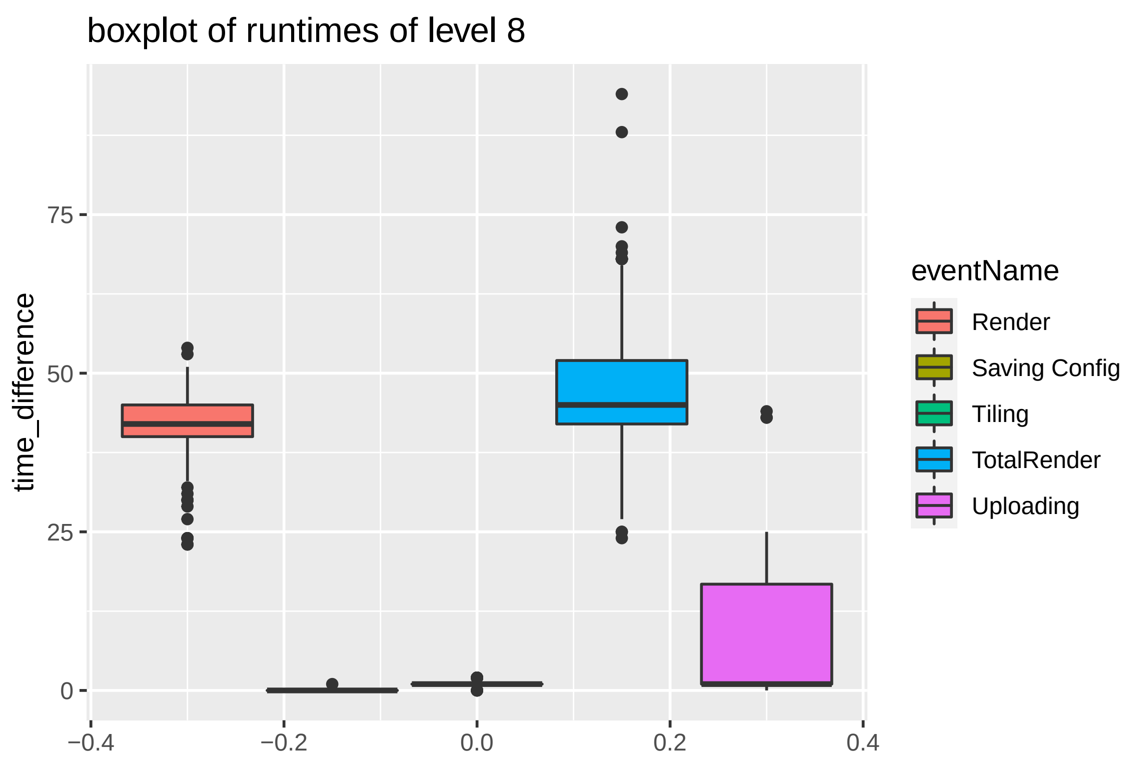


Figure 2: Analysis of desktop users based on total views of the modules



- with boxplot on events at level 8, it is very evident that render time dominates in total rendering runtime while uploading seems to have a significant outliers.
- other event names are as very small in comparison to rendering and uploading it seems.

2. Runtime analysis based for each event at level 12:

- Every event is filtered at level 12 and heat map of runtimes specific to tasks are plotted over x,y coordiantes as taskId has an unique x, y coordinates

- Most of the grid appears to have constant upload times except for a few areas of black dots representing the fastest upload times which seems to have a pattern or a trend as they are all located over a vertical line. this seems to be further analyzed to understand its nature.
- as per the above graphs represented, tiling and saving configuration times are almost constant for each taskId.
- In contrast to the total rendering and rendering time heat graphs, a map-like structure results from the scaling of color based on runtimes that resembles the supercomputer's tera pixel map. This creates a question in mind that whether the places where in a sq-area sensors are more, does it have some conncetion to less time difference in rendering process?, as it may have to work on a less area while compiling to each events. This may not give any specific significance to any analysis but the contrast of runtimes seems to be very much coexistent based on the depth of information that is to be processed by super computer - Here,top 10 outliers of highest runtimes are represented by black points on the graph. there is no specific significance to it yet.
- for more and significant understanding of runtimes, boxplot of each event is generated as below:

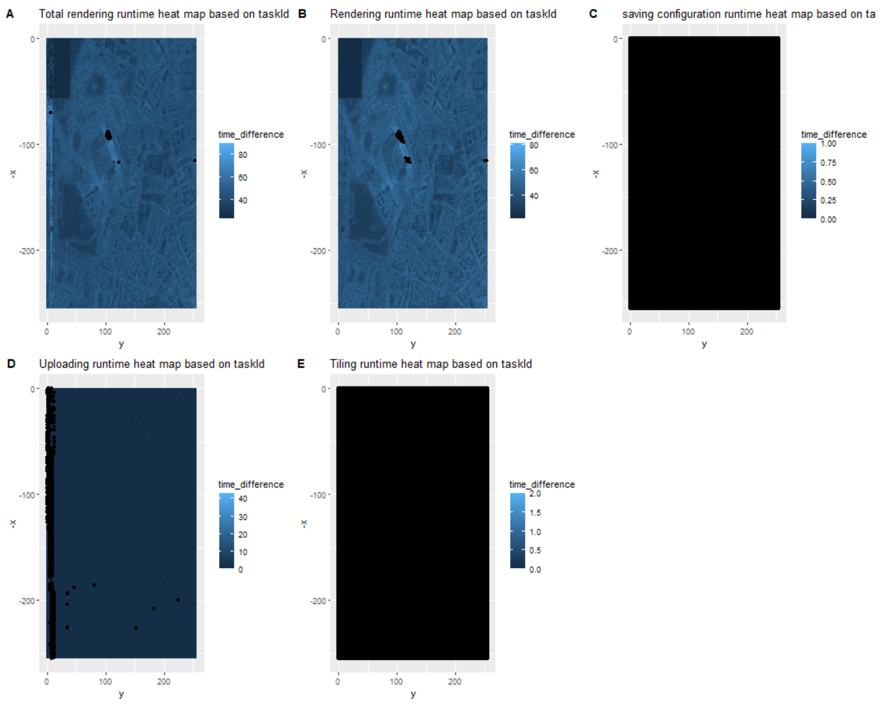


Figure 3: Analysis of desktop users based on total views of the modules

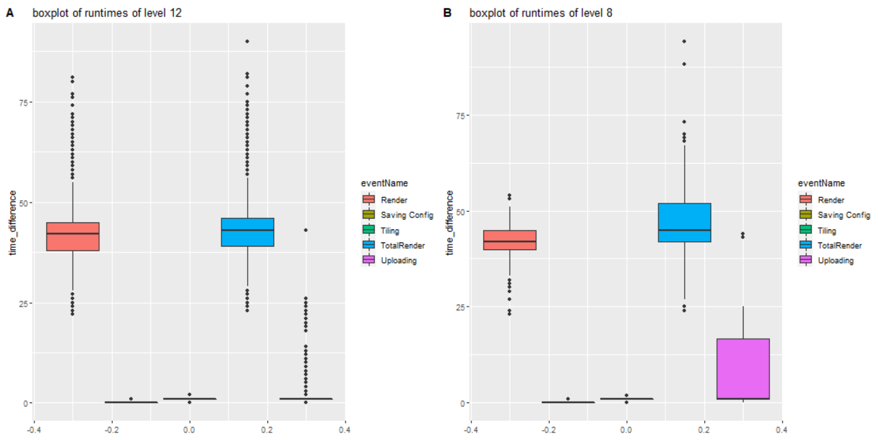


Figure 4: Analysis of desktop users based on total views of the modules

- According to the above boxplot, render is absolutely dominating the total rendering process. Also, there seems to be quite a significant number of outliers in uploading runtime. from the above heat map and boxplot of uploading runtime, it is very clear that there is an exceptional activity occurring which needs more information to understand what is happening.
- comparing with runtimes of level 12 and 8, uploading time looks to be more dominating in level 8 than that of level 12 with a similarity that uploading runtimes have many outliers.

##Performance analysis based on hostnames/GPU nodes

1. non idle hostname/gpu node performance

- non idle values of gpu nodes are summed and grouped by hostname to see how individual gpu nodes perform with the same under lines of idle and non idle values of gpu performance characteristics.
- After ordering the hostnames, they are indexed from 1 to 1024. Indexing has no bearing on the literature. The data can be plotted easily using indexes.
- Graphs showing non-idle host performance are shown below.

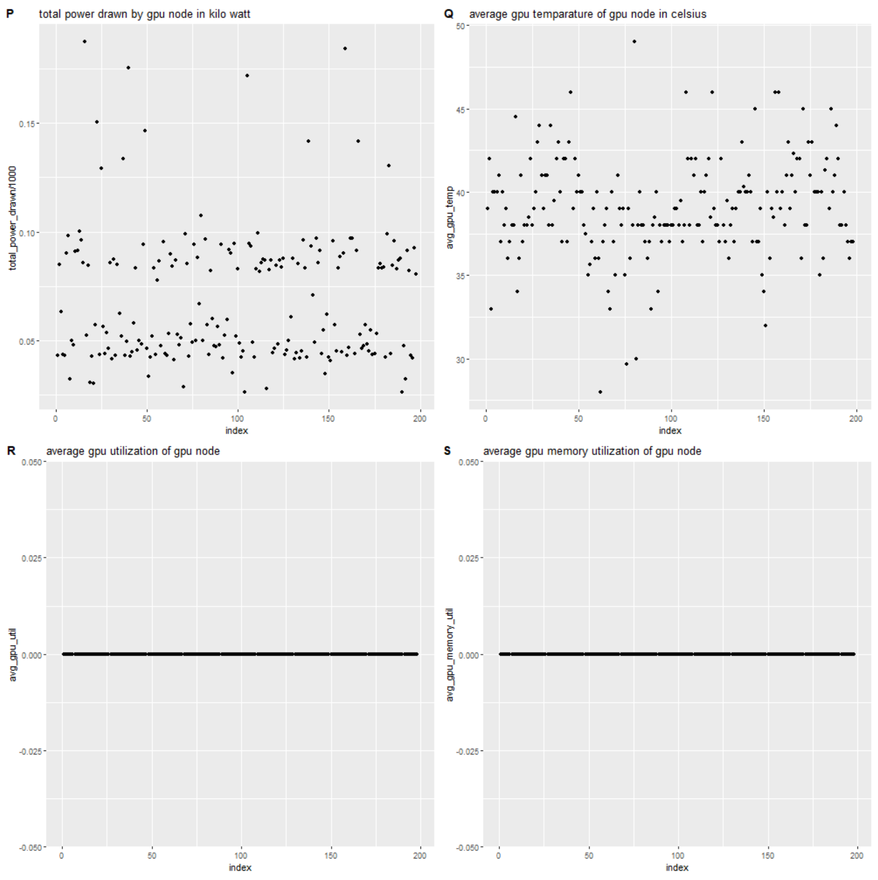


Figure 5: Analysis of desktop users based on total views of the modules

- According to the graphs above, the total power drawn by gpu nodes contains five outliers that drew much more power than the rest.

- For non-idela datasets, average gpu temperature is between 32 and 46 degrees Celsius; no precise interpretation can be deduced from other graphs because randomness across the index scale can be seen in other graphs, including the tempreture one.
- The following are the hostnames that drew a lot of electricity, as indicated by the dataset that was used to produce the total power drawn by the gpu node:

```
## [1] "0745914f4de046078517041d70b22fe7000005"
## [2] "35bd84d72aca403b8129a7d652cc2750000005"
## [3] "4a79b6d2616049edbf06c6aa58ab426a00000Y"
## [4] "4ad946d4435c42dabb5073531ea4f315000001"
## [5] "95b4ae6d890e4c46986d91d7ac4bf082000010"
```

2. Performance of idle hostnames/gpu nodes

- Similar to the preceding analysis, idle and non-idle values of gpu performance characteristics are summed and sorted by hostname to see how individual gpu nodes perform.
- Idle host performance graphs are shown below:

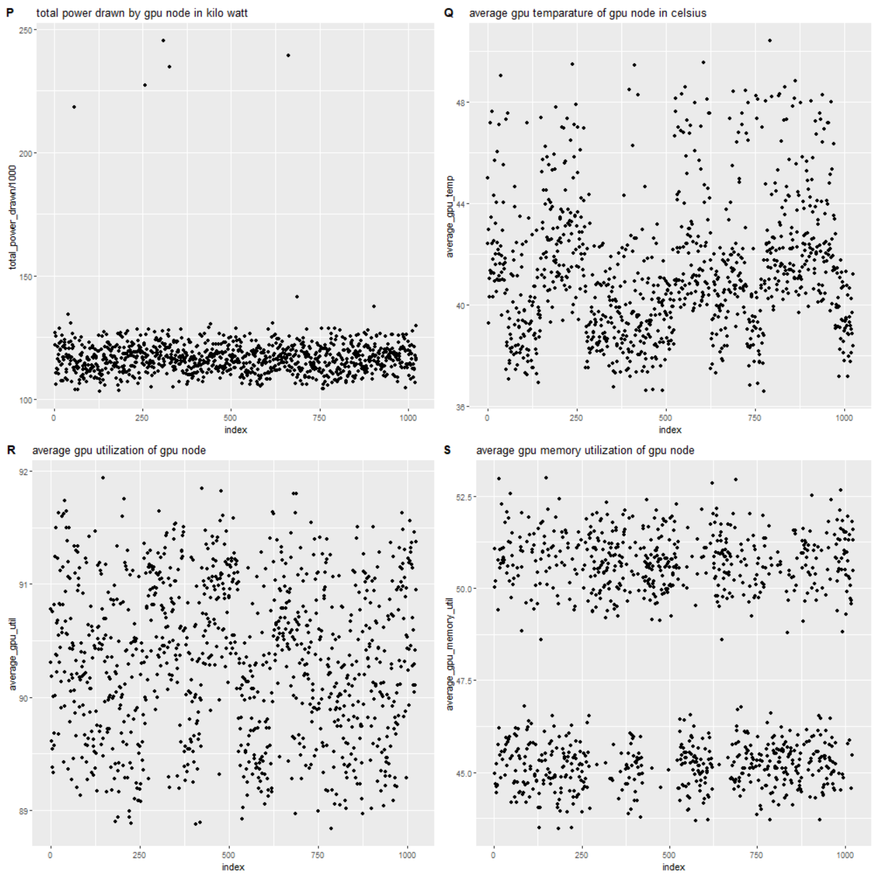


Figure 6: Analysis of desktop users based on total views of the modules

- According to the graphs above, total power drawn by gpu nodes include 5 outliers that drew much more power than the rest, which is comparable to what is shown in non-idle gpu node performance.

- In idle mode, gpu nodes utilise between 7 and 10 kilowatts of total electricity. Although the power consumed in idle state is modest in absolute terms, it does account for a certain amount of expenses that may be reduced when the idle state of gpu nodes is reduced.
- From the dataset used to generate total power drawn by gpu node, the following hostnames drew a lot of power when they were idle:

```
## [1] "0d56a730076643d585f77e00d2d8521a00000A"
## [2] "0d56a730076643d585f77e00d2d8521a000016"
## [3] "2ecb9d8d51bc457aac88073f6da0546100000E"
## [4] "83ea61ac1ef54f27a3bf7bd0f41ecaa7000009"
## [5] "cd44f5819eba427a816e7ce648adceb200000N"
```

- Because the outlier gpu nodes are separated into idle and non-idle cases, it's evident that the same gpu nodes are using a lot of power in both. This can be thought of as a focal point for replacing those specific gpu nodes in order to boost the super cloud computer's efficiency.

3.Using two colours to show idle vs non-idle GPU performance by hostnames in the same graphs.

- This graph will be more helpful in depicting the wide range of differences found in both gpu states:

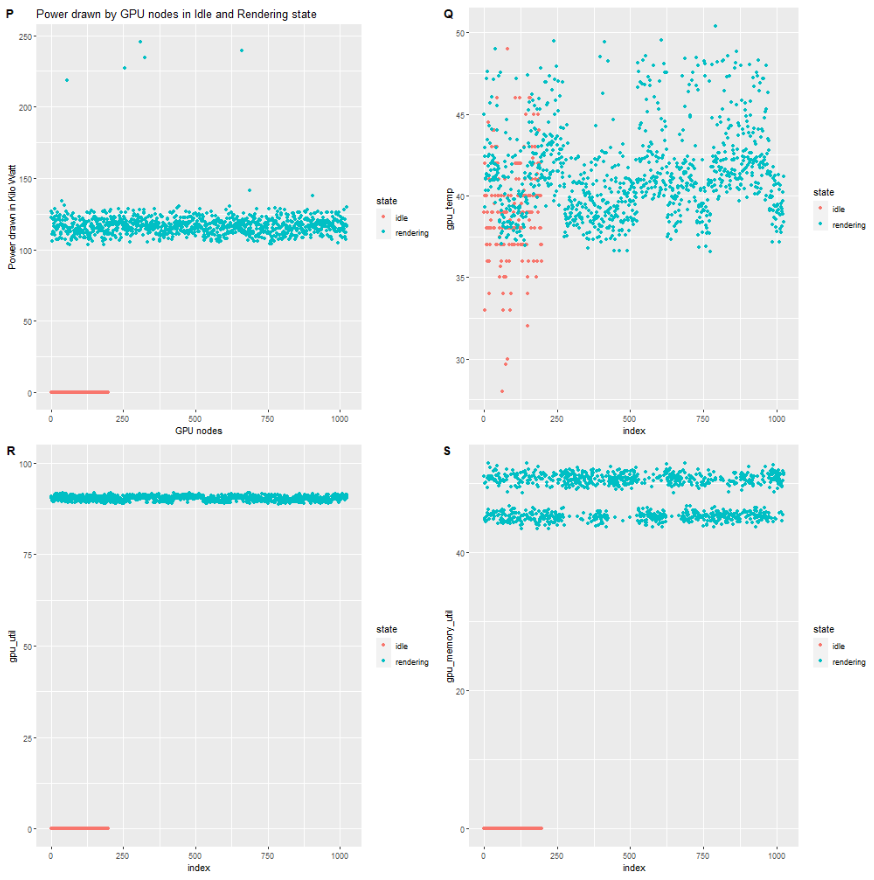


Figure 7: Analysis of desktop users based on total views of the modules

- Here it can be visualized effectively that how the total power drawn in the non-ideal state differs significantly from the consumed power in the ideal state; it also states that the ideal state increases the per-gpu temperature equivalent to the non-ideal performing gpus, and thus they may be hindering; and finally, the gpu memory utility can be seen to differ significantly in both states.

#Analysis of gpu variables on basis of seconds of day.

- GPU performance estimated with a filter on gpuUtil and gpuMemUtil larger than 10 can be deemed non-idle performance, as mentioned in data preprocessing step 8.
- GPU performance in time series is calculated to see how the supercomputer as a whole performs over time, therefore gpu variables are summarised in the same way as previously, grouped by seconds of the day.
- Time series graphs with seconds of the day plotted against summary GPU variables are shown below.

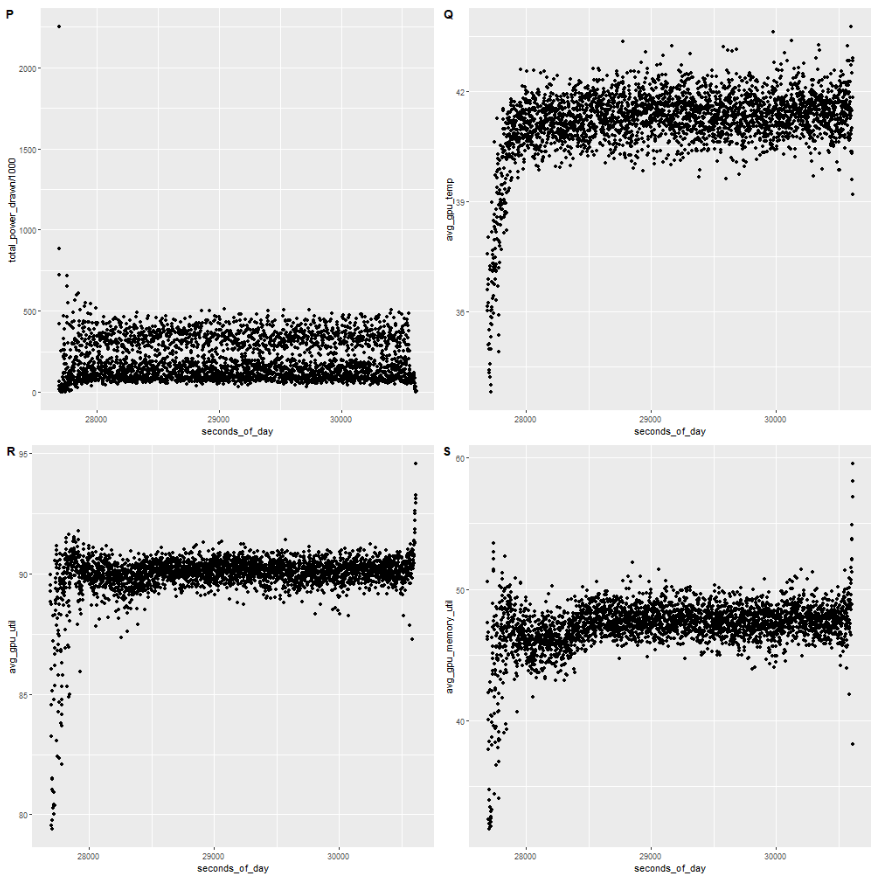


Figure 8: Analysis of desktop users based on total views of the modules

- When tera pixel rendering is started, the total power drawn by the supercomputer appears to be in the region of under 500 kilo Watts with few outliers, based on the above graphs. This could be linked to the uploading runtime anomaly.
- The temperature of the GPU rises over time until the rendering is completed.
- When rendering begins, gpu and memory utilisation appear to be at lower levels, however they fluctuate within the same range throughout the duration.

checking for-Idle performance time series examination of GPU variables:

- Idle values of variables are filtered by gpu utilisation & gpu memory utilisation less than 15, summarised by gpu variables, and grouped by seconds of day on the same premise as the above analysis.
- The time series graphs below show the supercomputer's overall performance when the GPU is idle :

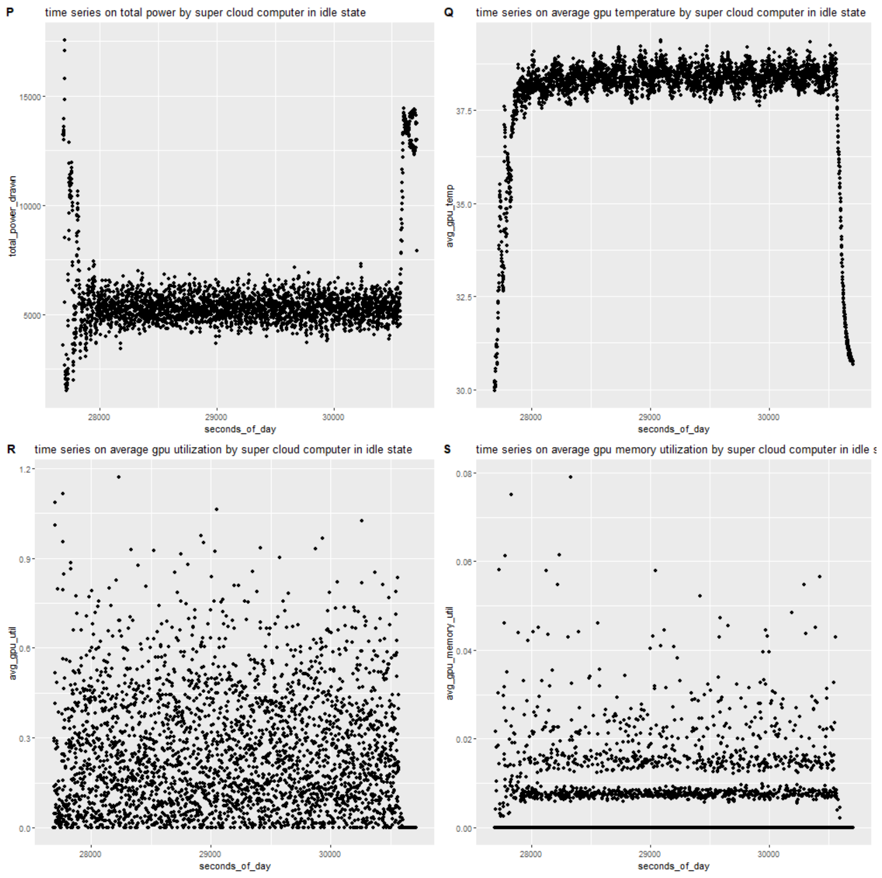


Figure 9: Analysis of desktop users based on total views of the modules

- According to the graphs above, the total power required by a supercomputer in idle mode is in the range of 4-7 kilowatts, but outliers can be seen at the beginning and conclusion of rendering a tera pixel. The reason for this behaviour cannot be discovered from this research, however it does serve as a starting point for further investigation.
- While gpu utilisation and gpu memory utilisation provide little useful information, a pattern of gpu temperature may be seen, as the temperature rises over time and then declines at the end of the process.
- In the end, this drastical change in temperature appears to be a more interesting topic of discussion and may lead to different results that may be useful, although it cannot be ellaborated here without more data on the matter.

Study of the variables that affect GPU performance

1. Level 12 gpu performance variables

- Power drawn in watts, gpu temperature in degrees Celsius, gpu utilisation, and gpu memory utilisation are all characteristics that might affect gpu performance.
- Each variable is summed and shown as a heat map using a unique taskId. While the total of powerWattDrawn is used to summarise the power drawn, the mean values are used to summarise the other variables.
- The heat maps below are sorted by taskId and based on aggregated gpu variables over x,y coordinates.

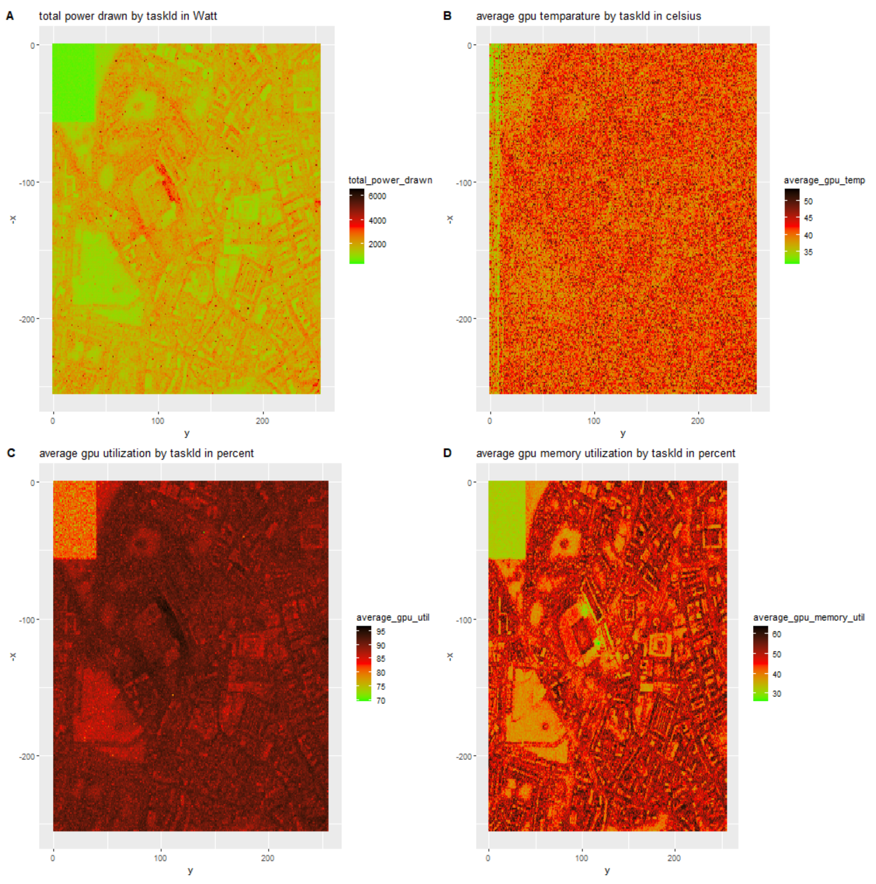


Figure 10: Analysis of desktop users based on total views of the modules

- It can be seen from the above graphs that the gpu temperature does not provide any information, whereas the other three heat maps resemble a map-like structure of the tera pixel rendered, and that it appears to be related to the number of sensors fixed in each area; as we look into the terapixel image, we can see that where the sensors are more placed, we can find less average power drawn and less average power memory utility.
- Although specific data cannot be understood, the heat map generated appears to be quite crisp, with map features plainly evident.

2. For level 8

- Gpu variables associated with level 8 rendering are summarized and plotted as below.

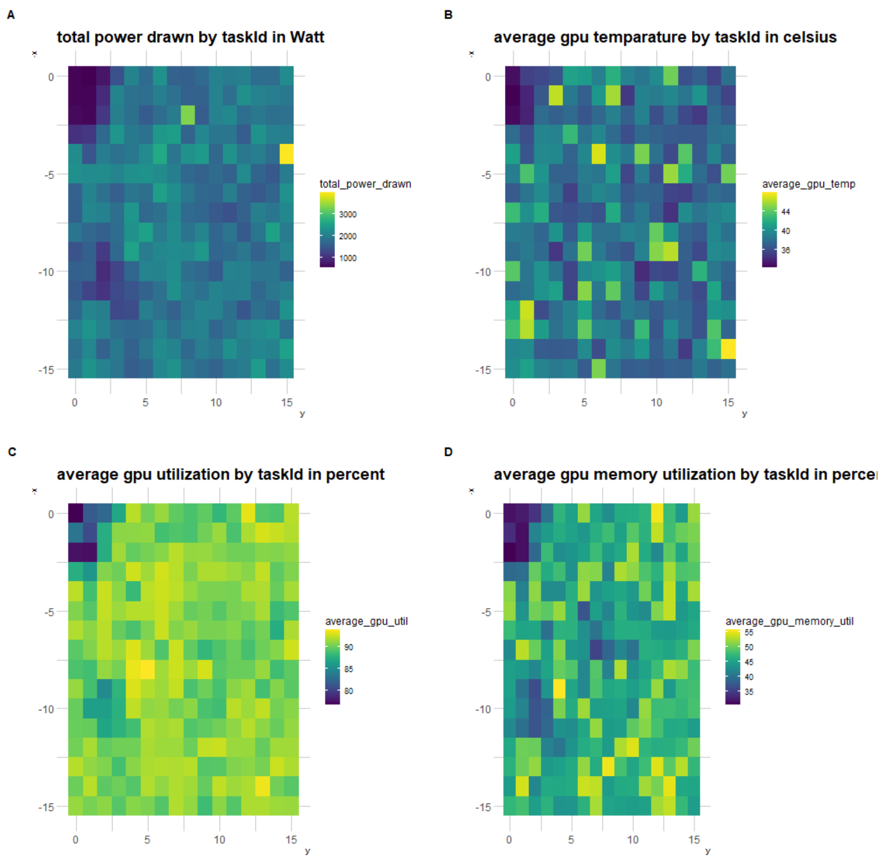


Figure 11: Analysis of desktop users based on total views of the modules

- no specific interpretation can be determined from the above heat maps, although the total power drawn by idel gpu is more in range from 1000-3000 and the average utility is on higher side i.e. here >=90.

Note:Here the above 2 analysis on gpu performance variables which generated heat maps are filtered on gpuUtil, gpuMemUtil whose values are greater than 15 as reasoned in data preprocessing step 8.

Analysis on total rendering time and gpu performance variables:

Interplay between increased power draw and total render time

- The total rendering time is a key number that represents the total run duration of any taskId, as well as how gpu variables fluctuate.
- On average power drawn, the complete rendering runtime is collected and summarised as time difference.
- The graph below were created using time difference drew drew aganist power drawn

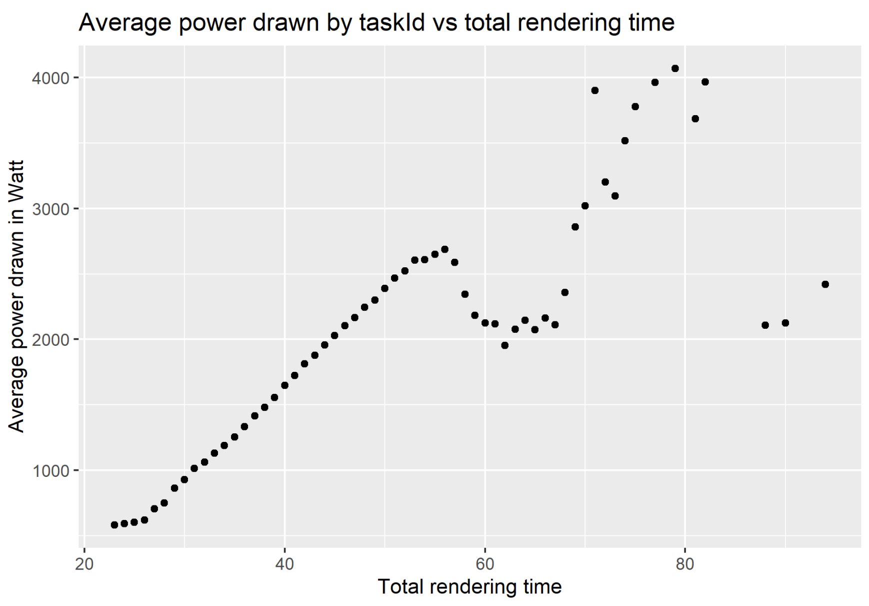


Figure 12: Analysis of desktop users based on total views of the modules

- As can be seen in the graph above, power drawn grows as total rendering time increases, but the pattern appears to alter between time difference 50 and 70, with a sharp reduction in all values that finally rose. Although no important reason has been discovered.
- USING CORRELATION TO CHECK OVERALL RELATION BETWEEN THIS 2 VARIABLES.

```
head(host_gpu_performance)
```

```
##                                   hostname powerDrawWatt gpuTempC gpuUtilPerc
## 1 04dc4e9647154250beeee51b866b0715000000         91.73       35          78
## 2 04dc4e9647154250beeee51b866b0715000000         85.79       35          79
## 3 04dc4e9647154250beeee51b866b0715000000         70.38       36          75
## 4 04dc4e9647154250beeee51b866b0715000000         65.90       36          81
## 5 04dc4e9647154250beeee51b866b0715000000         80.63       36          76
## 6 04dc4e9647154250beeee51b866b0715000000         88.60       36          77
##   gpuMemUtilPerc
## 1             29
## 2             31
## 3             30
## 4             31
## 5             34
## 6             31
```

```
# cor(x2$time_difference, x2$average_total_power_drawn)
```

- this relation confirms the linearity amount between two but the decrement after time diffrence 50 is still the question

On basis of plot-pattern obtained in above question,checking the pattern of other gpu variables with total render time:

- total rendering runtime stored as time_difference is grouped and summarized on gpu variables.
- below are graphs generated with time_difference aganist gpu variables

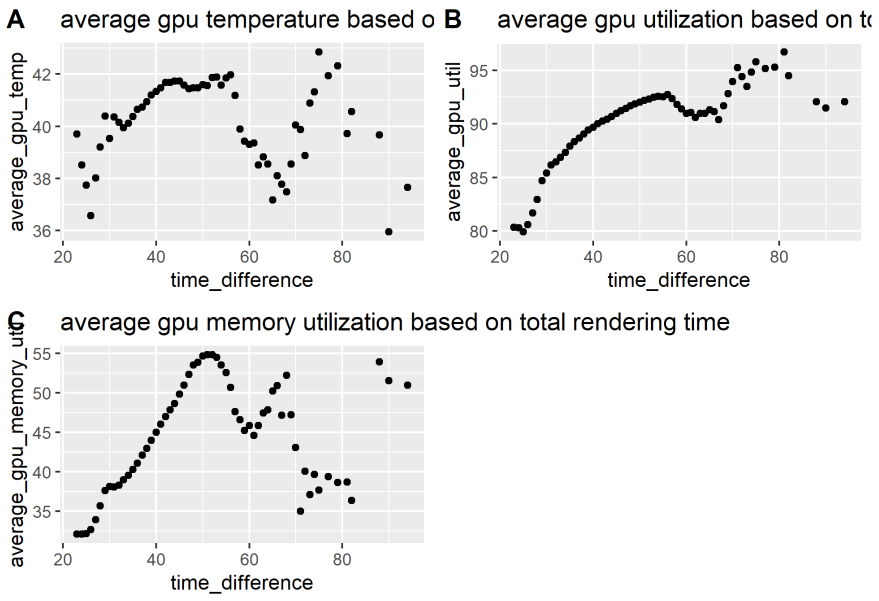


Figure 13: Analysis of desktop users based on total views of the modules

#Using Correlation method to check more about the relation between total render time vs gpu variables

```
#cor(x2$time_difference, x2$average_gpu_temp)
#cor(x2$time_difference, x2$average_gpu_memory_util)
#cor(x2$time_difference, x2$average_gpu_util)
```

- Similar pattern as power drawn graph is obatined here too,all gpu variables grow as total rendering time increases, however the trend appears to change route between time difference 50 and 70. All of the values dropped suddenly, but then rose again.

There was no significant reason found, but this can be used as a starting point to figure out why there is a decline in values when overall rendering time is really long.

Ideas and focal areas post analysis

- Operational approaches that might reduce gpu idle state, increasing super cloud computer availability and lowering power consumption.(Note : here we have considered gpu as idle if gpu utlization and gpu memory utilization are less than 15 percent)

1. How the sensors in a per square area affects the utility or consumption, further future research on it might be helpful in many ways.

2. Focus on ones having GPU utilisation and memory utilisation are less than 20%, work must be done investigating on it since they cost power, influence GPU temperature, and produce results that are not what we expected.
3. Introduce operating methods that could reduce idle state of gpu.

- In both idle and non-idle states,we found out GPU nodes that consume a lot of power,replace such outliers.
- Emphasis should be made on :

1. why is the average of all gpu variables dropping as total rendering time approaches 50 seconds?
2. why uploading runtimes are high at the start of terapixel rendering?
3. Idel state tempreture fluctuations are why decreasing in the final stages so rapidly and why in idle state total power drawn by super cloud computer is increasing in the initial and final stages of terapixel rendering.