Written Report

Cloud Computing - CSC8634

Muzaffer Senkal - 210351491

21/01/2022

1 Introduction

With the increase in IOT devices in recent years, digital twin technology plays an important role for companies.It is a virtual representation of the physical system [1]. It helps companies or organizations measure and visualize the performance of products in real time, as well as understand how they will perform in the future.

2 Business Understanding

The Newcastle Urban Observatory has been gathering environmental data from IoT devices which are located in Newcastle-upon Tyne. This data is presented in a browser by making a 3D visualization of the city. These visualization images are not simple image, they contain over one trillion pixels called terapixel images [2]. It allows viewers to interactively zoom into incredible detail on big data at multiple scales. On these images, the data received from the sensors such as temperature, humidity are shown live. Terapixel visualizations is is a crucial computational cost, so the university has used Microsoft Azure cloud services to perform the rendering process. Totally 1024 public cloud GPU nodes are used for this cost.

The terapixel image consists of (1,048,576)**2 pixels, and image tiles are 512x512 pixels in JPG format [2]. In this case, 5,592,405 image tiles will be created, and 581,610,120 kB storage will be needed. GPU nodes will compute 65,793 tasks.

2.1 Business objectives

It is very important to examine and monitor the performance of a running system. It determines the action to be taken so that the system can work better. It is useful not only to detect current performance issues, but also to identify other needs that may need improvement soon. Therefore, in this data mining project, the performance of the gpu-based system will be analyzed to answer the following questions;

- Which event types dominate task run times?
- What is the interplay between GPU temperature and performance?
- What is the interplay between increased power draw and render time?
- Can we quantify the variation in computation requirements for particular tiles?
- Can we identify particular GPU cards (based on their serial numbers) whose performance differs to other cards? (i.e. perpetually slow cards).
- What can we learn about the efficiency of the task scheduling process?

3 Data Understanding

3 data sets which are **Application Checkpoints**, **Gpu**, and **Task x.y** was created during a production of terapixel image using 1024 GPU nodes.These files are in well-know format which is comma-separated value

format. The data provided shows performance timing of the render application, performance of the GPU card, and details of which part of the image was being rendered in each task.

3.1 application-checkpoints.csv (111.2MB)

This file contains application checkpoint events throughout the execution of the render job. Examples of the events are given for the `eventName` field.

3.1.1 Schema

1. **timestamp:** time of data creation. Ex: `2018-11-08T07:41:55.921Z`
2. **hostname:** unique hostname created by the Azure batch system. Ex: `0d56a730076643d585f77e00d2d8521a00000N`
3. **eventName** Name of the event occuring within the rendering application.
    - **TotalRender** is the entire task
    - **Render** is when the image tile is is being rendered
    - **Saving Config** is simply a measure of configuration overhead
    - **Tiling** is where post processing of the rendered tile is taking place
    - **Uploading** is where the output from post processing is uploaded to Azure Blob Storage
4. **eventType:**
    - **START**
    - **STOP**
5. **jobId:** ID of the Azure batch job. Ex: `1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705`
6. **taskId:** ID of the Azure batch task. Ex: `b47f0263-ba1c-48a7-8d29-4bf021b72043`

3.1.2 Shape

```
dim(application.checkpoints)
```

```
## [1] 660400      6
```

3.1.3 Summary

Application checkpoints data includes 660400 rows and 6 columns. All columns represented as a character type. However, the types of some columns need to be changed before starting data analysis. For example, timestamp column should be represented by time format.

```
##   timestamp           hostname          eventName          eventType
##  Length:660400      Length:660400      Length:660400      Length:660400
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##     jobId              taskId
##  Length:660400      Length:660400
##  Class :character   Class :character
##  Mode  :character   Mode  :character
```

3.2 gpu.csv (208.7MB)

This file contains metrics that were output regarding the status of the GPU on the virtual machine.

3.3 Schema

1. **timestamp:** time of data creation. Ex: `2018-11-08T08:27:10.314Z`
2. **hostname:** unique hostname created by the Azure. Ex: `8b6a0eebc87b4cb2b0539e81075191b900001C`
3. **gpuSerial:** The serial number of the physical GPU card.Ex: "0323217055910"
4. **gpuUUID:** The unique system id assigned by the Azure system to the GPU unit. Ex: `GPU-1d1602dc-f615-a7c7-ab53-fb4a7a479534`
5. **powerDrawWatt:** Power draw of the GPU in watts. Ex: `131.55`
6. **gpuTempC:** Temperature of the GPU in Celsius Ex: `48`
7. **gpuUtilPerc:** Percent utilisation of the GPU Core(s). Ex: `92`
8. **gpuMemUtilPerc:** Percent utilisation of the GPU memory. Ex: `52`

3.3.1 Shape

```
dim(gpu)
```

```
## [1] 1543681       8
```

3.3.2 Summary

Gpu data set consist of 1048575 rows and 8 columns. timestamp, hostname, and gpuUUID columns are listed as a character and the remains are represented as numerical.

```
##   timestamp           hostname           gpuSerial           gpuUUID
##  Length:1543681     Length:1543681     Min.   :3.201e+11   Length:1543681
##  Class :character   Class :character   1st Qu.:3.236e+11   Class :character
##  Mode  :character   Mode  :character   Median :3.236e+11   Mode  :character
##                                        Mean   :3.240e+11
##                                        3rd Qu.:3.250e+11
##                                        Max.   :3.252e+11
##  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
```

3.4 task-x-y.csv (6.2MB)

This file contains the x,y co-ordinates of which part the image was being rendered for each task.

3.4.1 Schema

1. **jobId:** Id of the Azure batch job. Ex: `1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705`
2. **taskId:** ID of the Azure batch task. Ex: `b47f0263-ba1c-48a7-8d29-4bf021b72043`
3. **x:** X co-ordinate of the image tile being rendered. Ex: `116`
4. **y:** Y co-ordinate of the image tile being rendered. Ex: `178`
5. **level:** Visualization output, zoom level

- **4**
- **8**
- **12**

3.4.2 Shape

```
dim(task.x.y)
```

```
## [1] 65793     5
```

3.4.3 Summary

This data contains 65793 observations with 5 columns. taskId and jobId columns are listed as a character type, and the rest of columns are numeric.

```
##     taskId             jobId                x             y
##  Length:65793       Length:65793       Min.   :  0   Min.   :  0
##  Class :character   Class :character   1st Qu.: 63   1st Qu.: 63
##  Mode  :character   Mode  :character   Median :127   Median :127
##                                        Mean   :127   Mean   :127
##                                        3rd Qu.:191   3rd Qu.:191
##                                        Max.   :255   Max.   :255
##      level
##  Min.   : 4.00
##  1st Qu.:12.00
##  Median :12.00
##  Mean   :11.98
##  3rd Qu.:12.00
##  Max.   :12.00
```

3.5 Data Quality

3.5.1 Missing Values

Missing data can significantly affect our analysis. But when we check the 3 data sets, it is clear that there is no missing data.

```
sum(is.na(gpu))
```

```
## [1] 0
```

```
sum(is.na(application.checkpoints))
```

```
## [1] 0
```

```
sum(is.na(task.x.y))
```

```
## [1] 0
```

3.5.2 Duplicate Rows

```
nrow(gpu) - gpu %>% distinct() %>% nrow()
```

```
## [1] 9
```

```
nrow(application.checkpoints) - application.checkpoints %>% distinct() %>% nrow()
```

```
## [1] 2470
```

```
nrow(task.x.y) - task.x.y %>% distinct() %>% nrow()
```

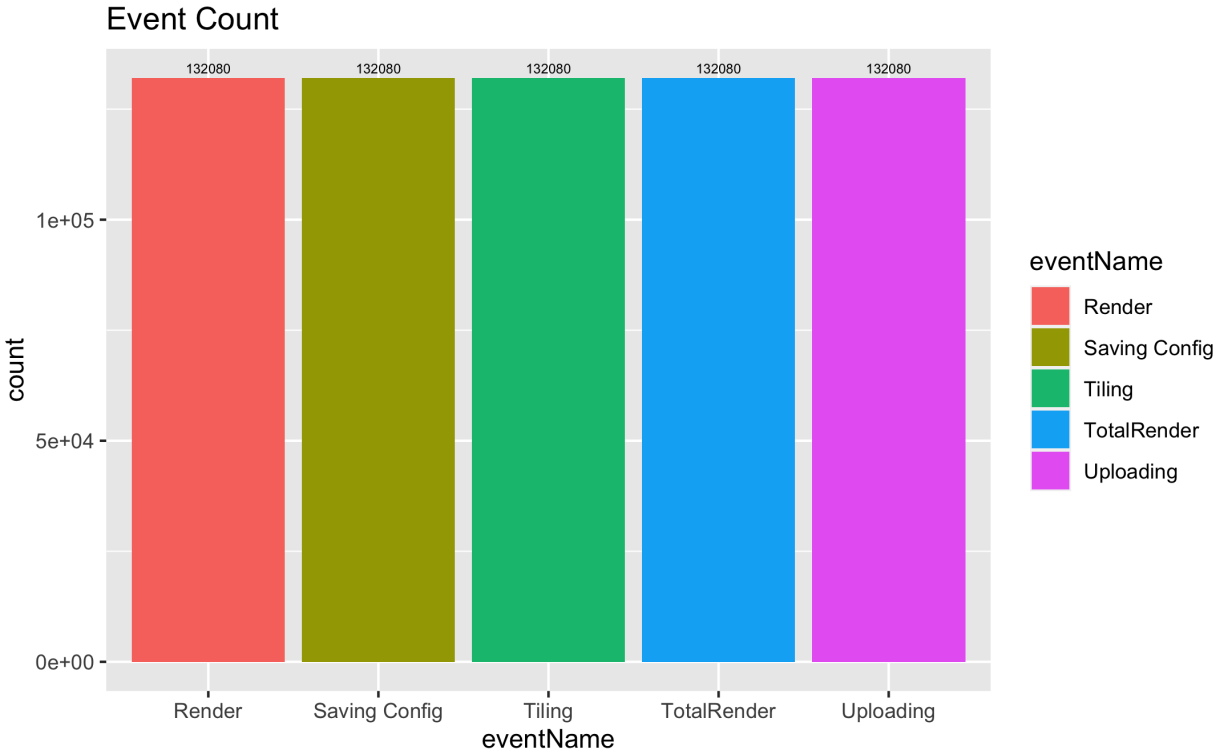
```
## [1] 0
```

There are 2.470 rows are duplicated in application.checkpoint data. We should consider in our analysis. In the gpu data frame , there are only 9 rows duplicated. However, task.x.y data frame does not contain duplicate rows.

3.6 Exploratory data analysis

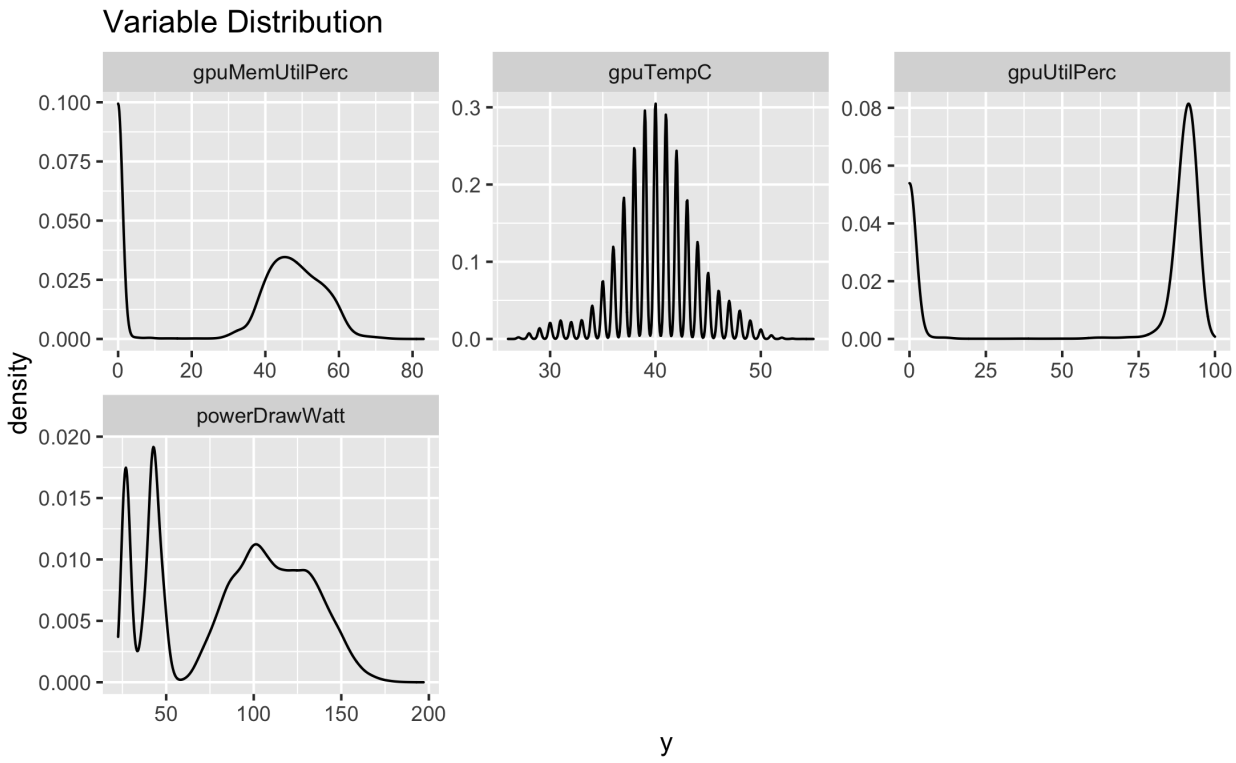
3.6.1 Event Count

According to the bar graph, event types are all equal which is 132080. There is no missing or extra records in the data.



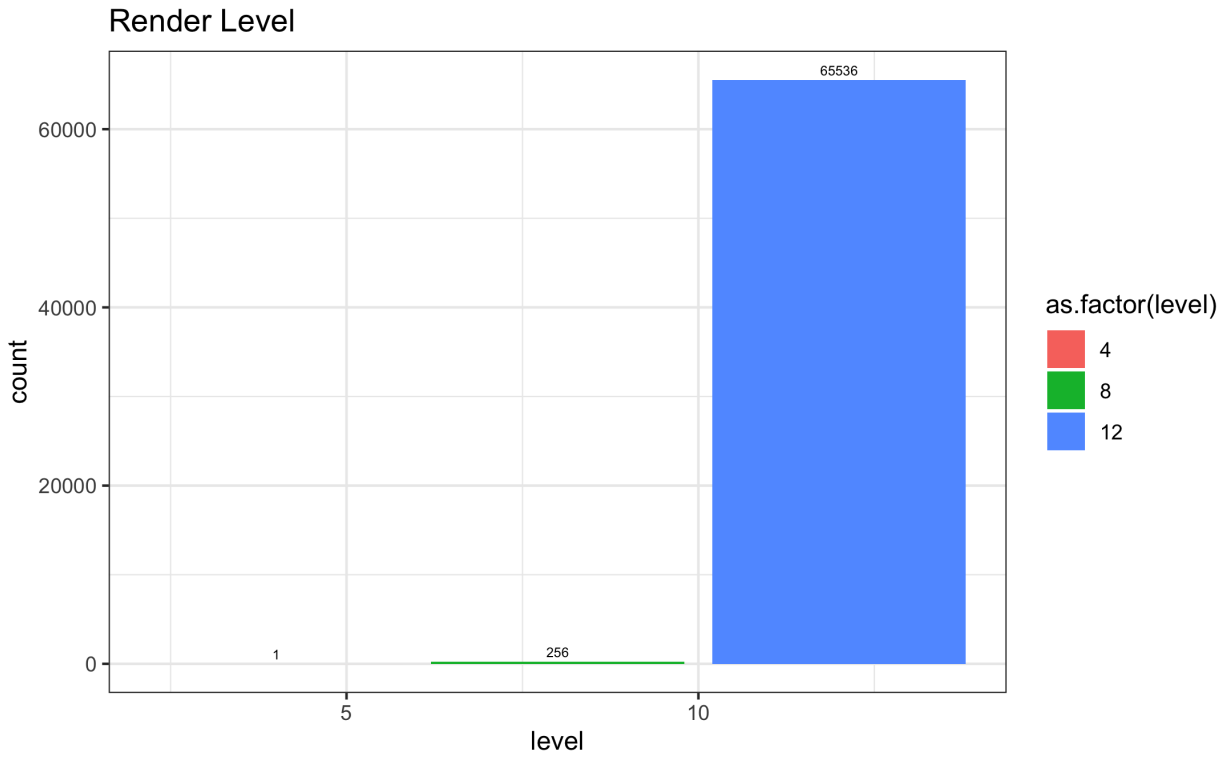
3.6.2 Variable Distribution

Looking at the graph below, the first thing to notice is that the GPU temperature is in a normal distribution. The mean of the temperature looks 40 Celcius. GPU utilization percentage is between 75 and 100, and also GPU usage drops below 10%. GPU usage may differ between event types. The power usage of the GPU is up to 200.



3.6.3 Render Level

As stated in the business understanding section, there are 3 render levels, and 65,793 tasks. The level that requires the most render is 12.



3.6.4 GPU Unique Values

gpuSerial, hostname and gpuUUID have the same number of unique values.

```
print(paste0("Count of Unique gpuSerial Values: ", length(unique(gpu$gpuSerial))))
```

```
## [1] "Count of Unique gpuSerial Values: 1024"
```

```
print(paste0("Count of Unique hostname Values: ", length(unique(gpu$hostname))))
```

```
## [1] "Count of Unique hostname Values: 1024"
```

```
print(paste0("Count of Unique gpuUUID Values: ", length(unique(gpu$gpuUUID))))
```

```
## [1] "Count of Unique gpuUUID Values: 1024"
```

3.6.5 Application Checkpoint Unique Values

The number of tasks required to produce a terapixel image is 65793. According to the information below, there are 3 jobIds. These are unique values assigned by Azure for each render level.

```
print(paste0("Count of Unique hostname Values: ", length(unique(application.checkpoints$hostname))))
```

```
## [1] "Count of Unique hostname Values: 1024"
```

```
print(paste0("Count of Unique jobId Values: ", length(unique(application.checkpoints$jobId))))
```

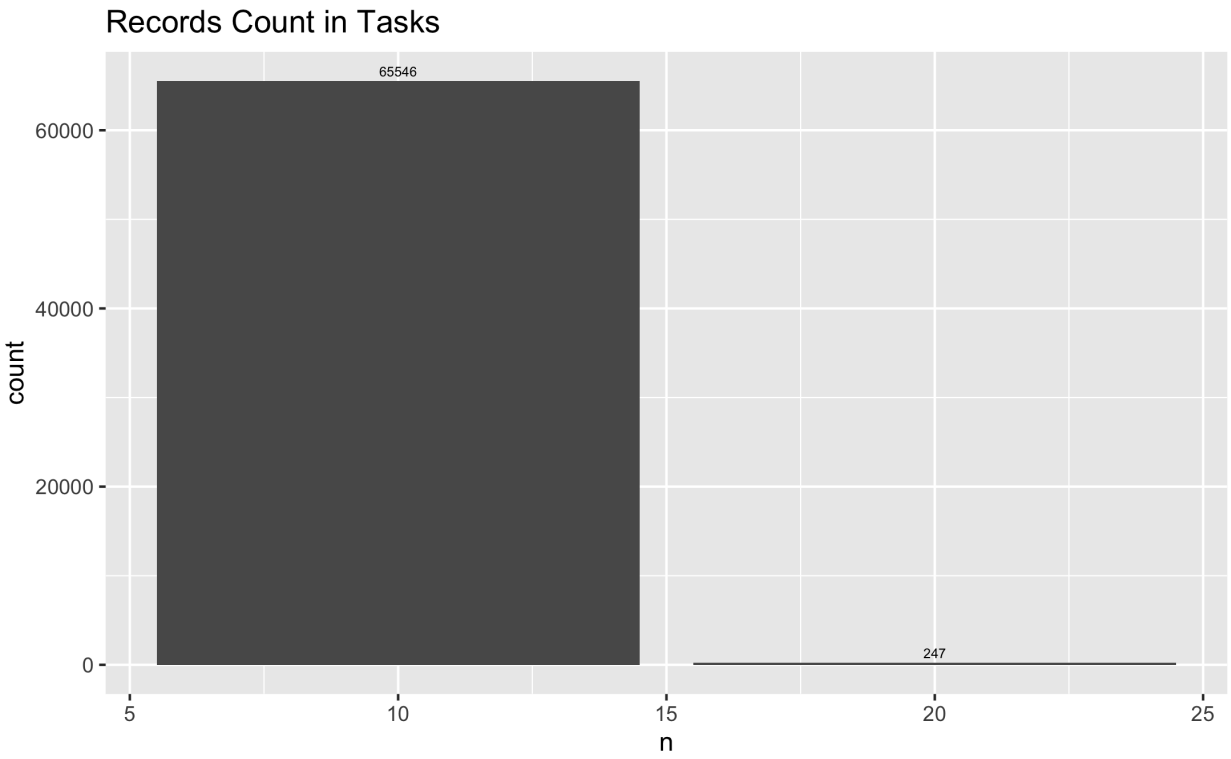
```
## [1] "Count of Unique jobId Values: 3"
```

```
print(paste0("Count of Unique taskId Values: ", length(unique(application.checkpoints$taskId))))
```

```
## [1] "Count of Unique taskId Values: 65793"
```

3.6.6 Records Count in Tasks

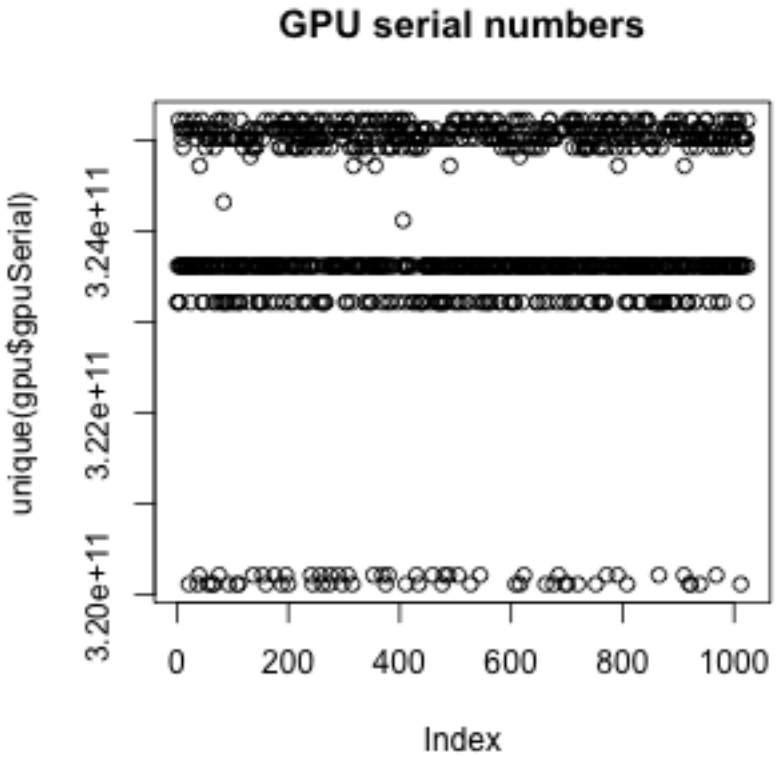
5 events are required to render an image tile. And these events are expected to be 10 in total with start and end records. However, 20 records appear for some tasks. This indicates that some records are duplicates.



3.6.7 GPU Serial

There is no information about the models of the graphics cards on the data. But the GPU serial numbers contain the manufacturing information of cards. For example:

Serial no: 07SB1609001016 **Description:** Within the serial number after the letter B, the 16 represents the year and the 09 represents the month of manufacture [3]. In this example, this VGA card was manufactured on September 2016.



It is clear that there is a pattern in serial numbers. We can distinguish the types of GPU cards from these numbers.

4 Data Preparation

Data from the three files provided to us will be used to evaluate the supercomputer creation process. In this part, we will covers all process to construct final data from raw data in order to prepare the data for further analysis.

4.1 Machine Unique ID

There are 1024 virtual machines in total in the system, and all of the machines have a single GPU. In order to better understand the data and make visualization easier, we will assign a unique id called **machine_id** from 1 to 1024 for each machine.

4.2 Delete duplicate rows

As we stated in the data understanding part, 2,470 duplicates found for **gpu** and 9 duplicates for **application.checkpoints**. These values will be deleted to avoid problems in the analysis.

4.3 Merge Data with Task.x.y

In the next steps, we will need to analyze the x and or coordinates of the therapeutic image. For this reason, Task.x.y data set merged into application.tasks.

4.4 Convert Date Format

In the data understanding section, it was mentioned that the "timestamp" column is of character type. To calculate elapsed time for event types, "timestamp" column should be in **POSIXct** type.

4.5 Calculate Elapsed Time

There is no total elapsed time information for each activity in the data. But there are start and end times in **application.checkpoints**. Thus, this information can be obtained by finding the difference between this start and end time. We will store the new data in the variable as **application.events**. Additionally, the elapsed time of each task was calculated and stored in a new data frame called **application.tasks**.

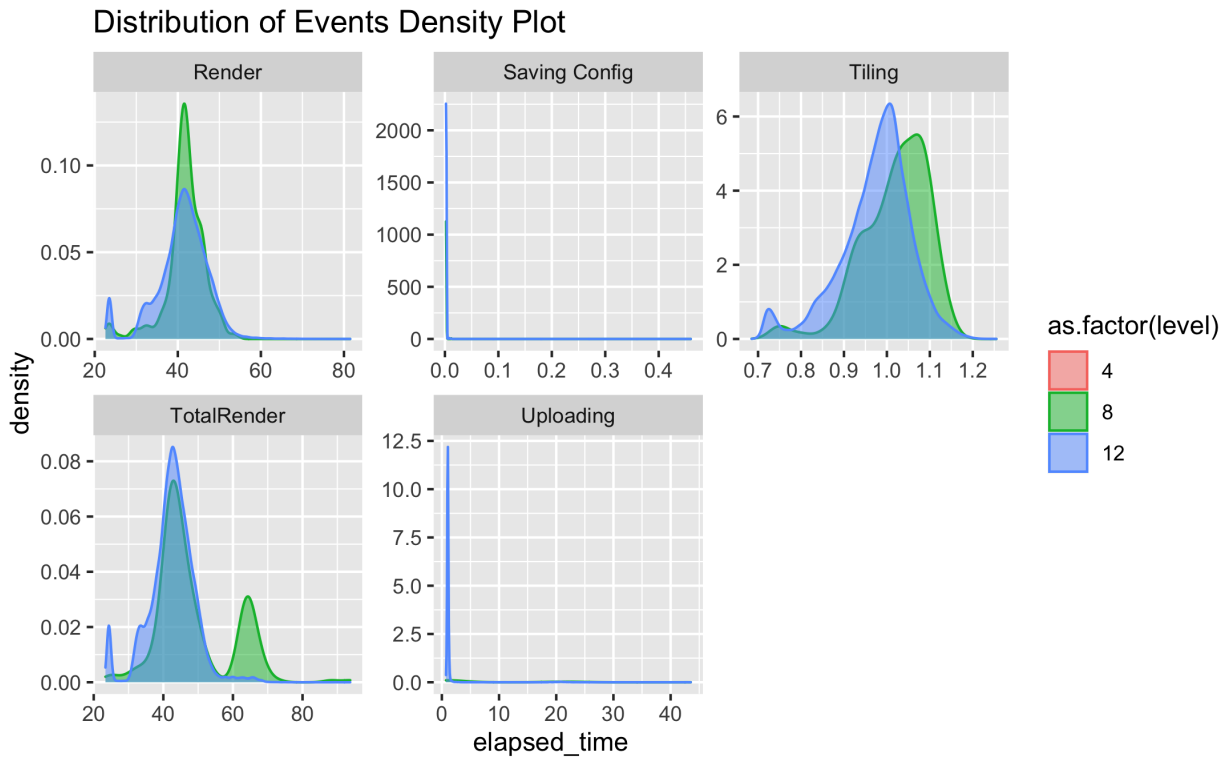
5 Modelling

Analyses of the data that we have obtained and processed will be conducted in order to achieve the aim and answer the questions raised in the previous section of the report. The analysis would be organised according to the order of the questions listed in the report's Business Objective section.

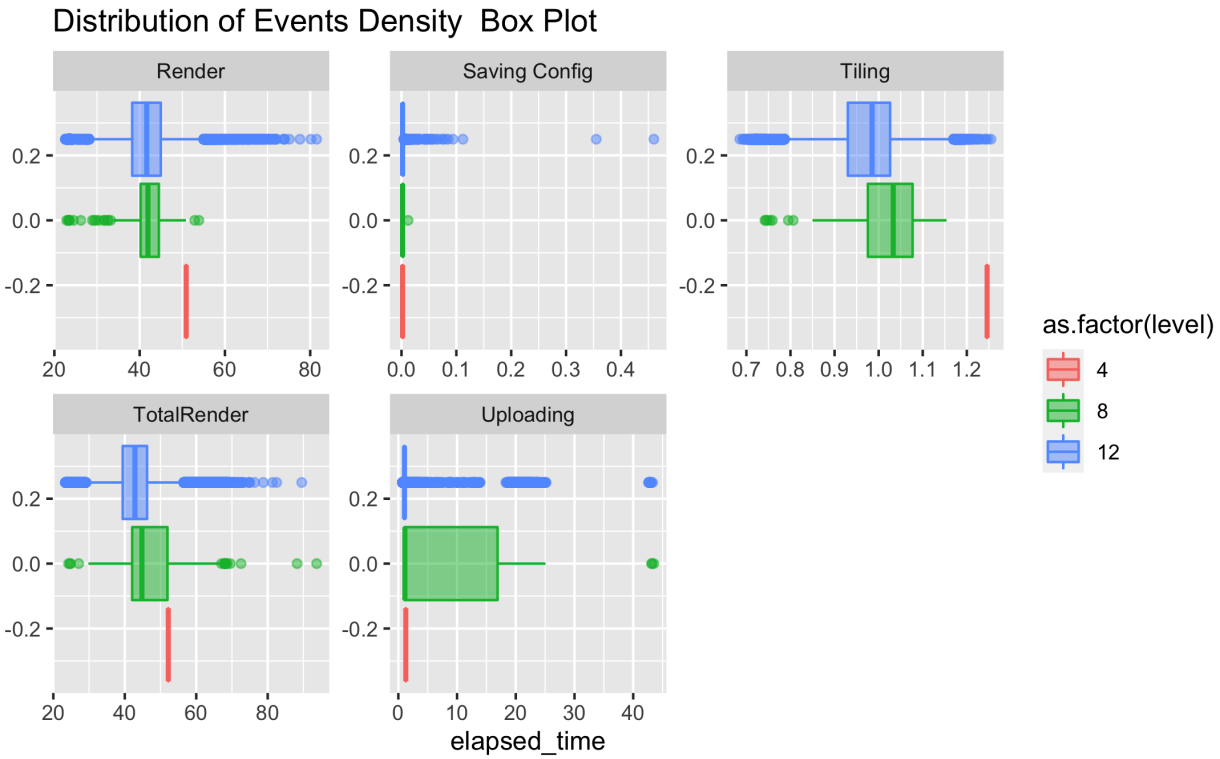
5.1 Which event types dominate task run times?

Application.events data will be used to determine which event is dominating the task run time. In the data preparation step, the time taken for each activity was calculated, and assigned to a variable called **application.events**. A box plot and density plot would be plotted to demonstrate the spread and mean of each event name duration to answer the question of which event dominates the event run time. In addition, these graphics will be grouped according to the level of visualization. Because the render time of each level can be different.

5.1.1 Density Plot



5.1.2 Box Plot



The **Total Render** event should be excluded as it represents the entire task run time. Therefore, it is evident from the graphs above that the **Render** event type dominates the event execution time. In general, regardless of the visualization level, when the running times of the activities are compared, the Render event covers **94.5%** of the total time while producing the terapixel image.

```
## # A tibble: 4 x 3
##       X eventName      percent
##   <int> <chr>            <dbl>
## 1     1 Render         94.6
## 2     2 Saving Config   0.00568
## 3     3 Tiling          2.23
## 4     4 Uploading       3.20
```

5.2 What is the interplay between GPU temperature and performance?

We'll use **gpu** data and **application.tasks** to answer the question about the relationship between GPU temperature and GPU performance. Generally, the performance of a GPU is assessed by how well it renders an object. However, there is no information about the quality of the tile of image. Therefore, we'll assume every tile is equal and evaluate GPU performance in terms of render time.

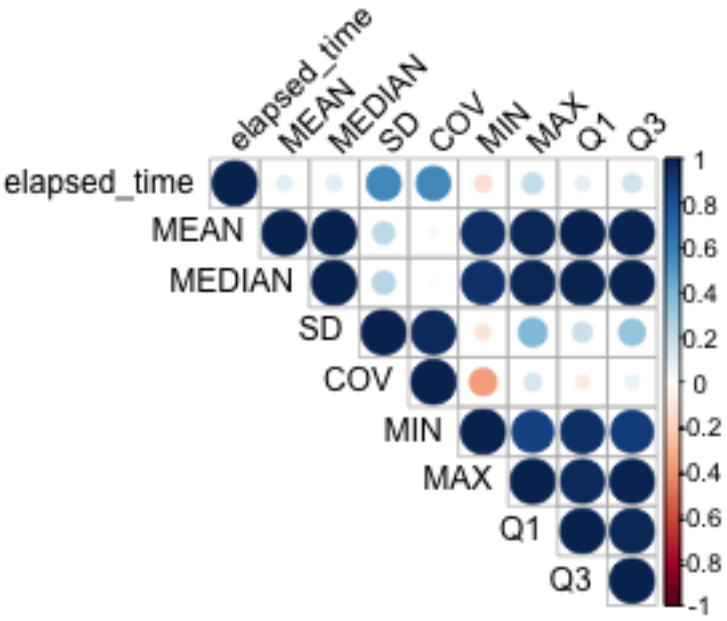
We have data about the render time for each tile. For each of these tiles, we will get the temperature values from the start and end time intervals of the tiles in the GPU data. Thus, we will obtain temperature data series for each render time. We'll look at the relationship between the elapsed time and GPU temperature during this process to see if GPU temperature has an impact on GPU performance. To investigate the correlation, we will use temperature values such as **mean,median, standard deviation, maximum, and minimum, coefficient of variation, quantiles**.

First, the table that correlates the temperature data and the elapsed time of the tiles needs to be created.

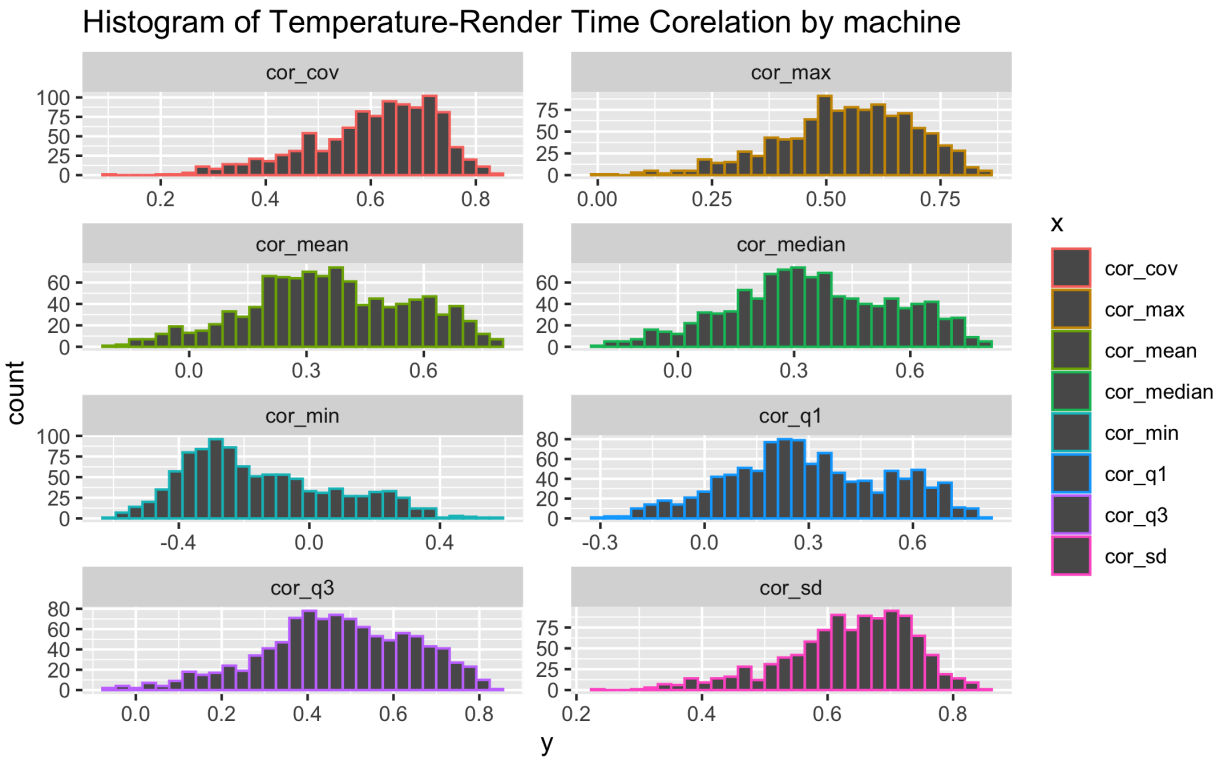
```
## # A tibble: 9 x 10
```

```
##   X      elapsed_time    MEAN  MEDIAN     SD     COV    MIN   MAX      Q1     Q3
##   <chr>         <dbl>   <dbl>   <dbl>  <dbl>   <dbl>  <dbl> <dbl>   <dbl>  <dbl>
## 1 elaps~        1       0.104   0.107   0.568  0.565  -0.134 0.206  0.0901 0.167
## 2 MEAN          0.104   1       0.995   0.226 -0.0357  0.935 0.978  0.992  0.990
## 3 MEDIAN        0.107   0.995   1       0.246 -0.0147  0.921 0.974  0.988  0.985
## 4 SD            0.568   0.226   0.246   1      0.962  -0.112 0.392  0.181  0.335
## 5 COV           0.565  -0.0357 -0.0147  0.962  1      -0.363 0.141 -0.0809 0.0791
## 6 MIN          -0.134   0.935   0.921  -0.112 -0.363   1     0.858  0.938  0.888
## 7 MAX           0.206   0.978   0.974   0.392  0.141   0.858 1      0.960  0.986
## 8 Q1            0.0901  0.992   0.988   0.181 -0.0809  0.938 0.960  1      0.975
## 9 Q3            0.167   0.990   0.985   0.335  0.0791  0.888 0.986  0.975  1
```

According to the correlation table, there does not appear to be a strong correlation between task duration and temperature attributes,but the elapsed time has a positive correlation of **0.56** with the temperature standard deviation and coefficient of variation. It means that how much temperature values are spread out around the mean or average affect elapsed time.



On the other hand, the performance and capacities of the 1024 GPU cards in the system may differ. Collective evaluation can be misleading. Therefore, a comparison between temperature and performance should be made per GPU.



According to the histogram above, the correlation coefficients between the elapsed time and the standard deviation of the temperature values spread approximately between 0 and 0.85. Peak coefficients range from about 0.6 to 0.75, and the histogram is slightly skewed to the left. It can be said that there is a small positive correlation. This interpretation will also be same for the coefficient of variance. The 2 histograms are very similar to each other. Furthermore, there is some correlation between the maximum temperature and the elapsed time. Peak values appear above but,t is not a strong correlation. Looking at other metrics, it seems that the correlation coefficients are distributed over small values.

5.3 What is the interplay between increased power draw and render time?

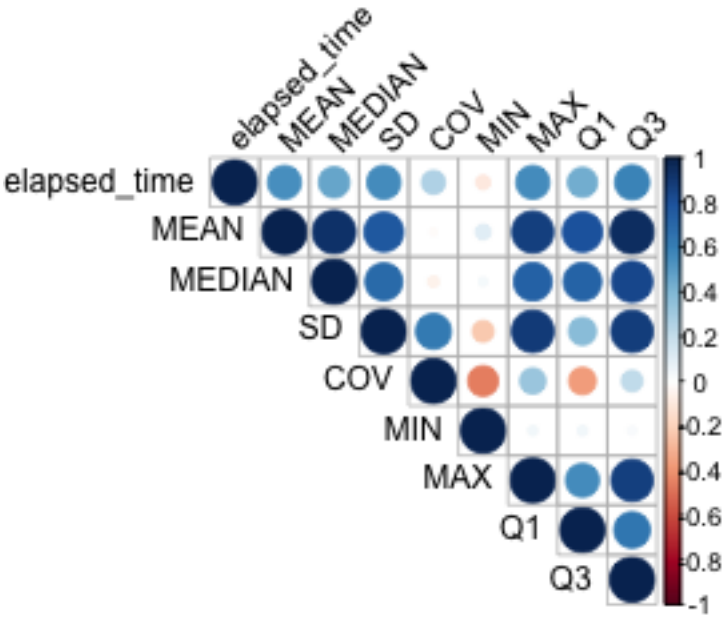
To carry out this analysis, the methods performed in the previous stage will be followed. The correlation between the data in the **Power Draw** column in the GPU data and the elapsed time will be examined.

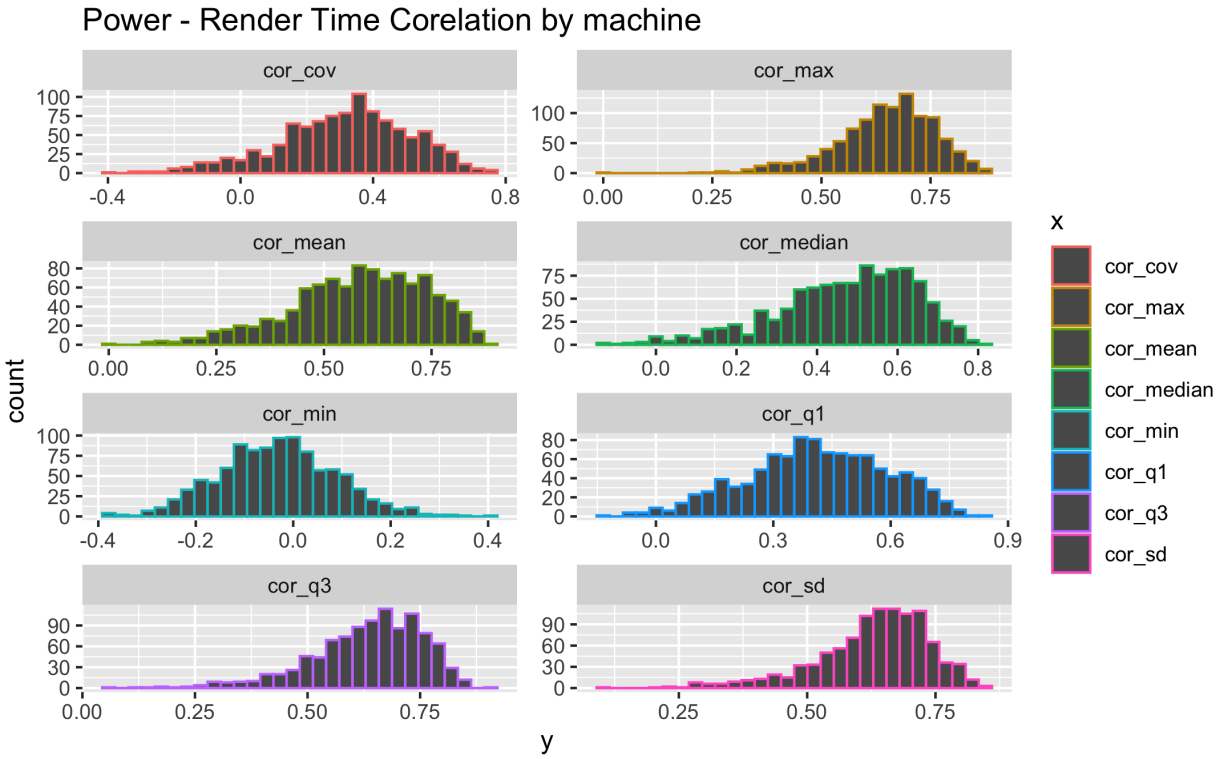
```
df_power_cor
```

```
## # A tibble: 9 x 10
##   X            elapsed_time    MEAN  MEDIAN     SD     COV     MIN    MAX      Q1
##   <chr>               <dbl>   <dbl>   <dbl>  <dbl>   <dbl>   <dbl>  <dbl>   <dbl>
## 1 elapsed_time        1       0.550   0.468   0.556  0.269  -0.0957 0.550   0.433
## 2 MEAN                0.550   1       0.924   0.767 -0.0199  0.109  0.863   0.790
## 3 MEDIAN              0.468   0.924   1       0.688 -0.0571  0.0333 0.720   0.716
## 4 SD                  0.556   0.767   0.688   1      0.616  -0.220  0.882   0.373
## 5 COV                 0.269  -0.0199 -0.0571  0.616  1      -0.451  0.326  -0.361
## 6 MIN                -0.0957  0.109   0.0333 -0.220 -0.451   1      0.0402  0.0434
## 7 MAX                 0.550   0.863   0.720   0.882  0.326   0.0402 1       0.552
## 8 Q1                  0.433   0.790   0.716   0.373 -0.361   0.0434 0.552   1
## 9 Q3                  0.587   0.940   0.832   0.876  0.218   0.0296 0.867   0.639
## # ... with 1 more variable: Q3 <dbl>
```

According to the correlation table, many temperature statistics seem to have a correlation of more than 0.5 with elapsed time. These temperature statistical values are as follows; **mean, standart deviation, max,

third quantile*. However, correlations are not strong.





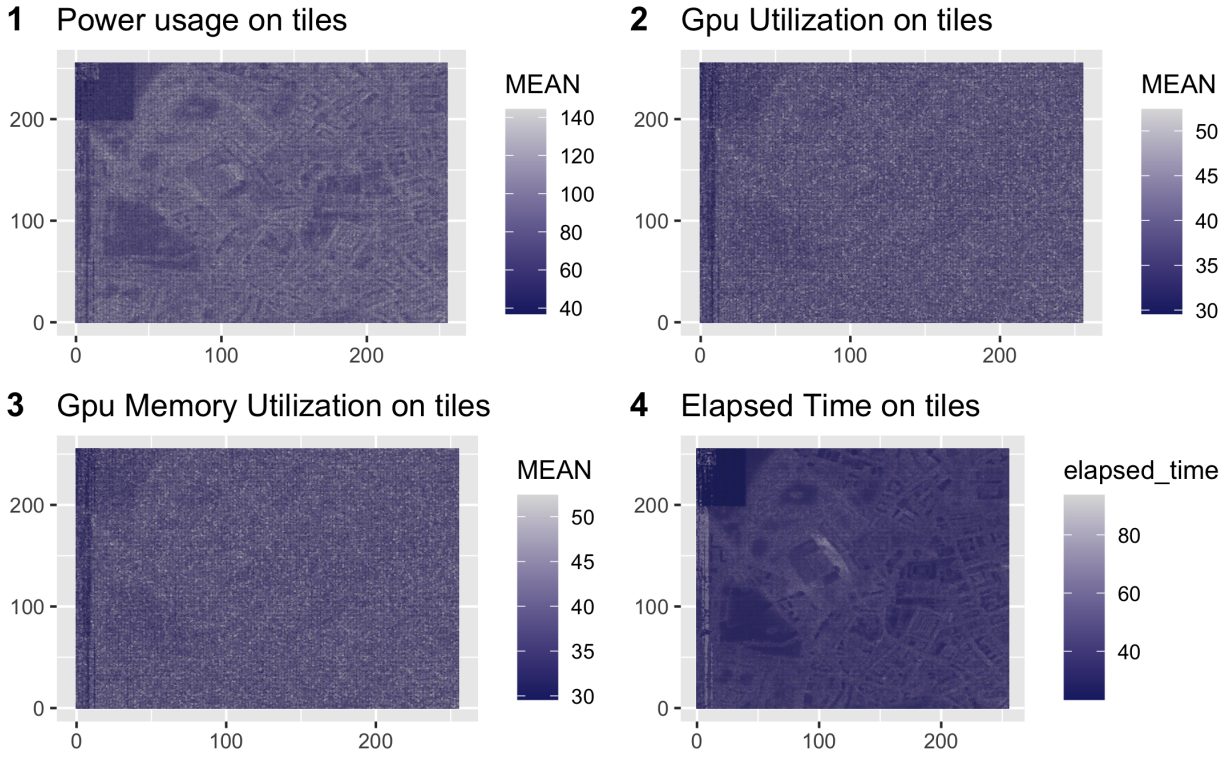
According to the histogram above, the correlation coefficients between the elapsed time and the standard deviation of the power draw peak values spread approximately between 0.60 and 0.725, and slightly skewed to the left. The correlation coefficients of the maximum and median values also seem to be quite high. It seems that there is a positive correlation between performance and performance in the third quarter.

5.4 Can we quantify the variation in computation requirements for particular tiles?

Each tile has different rendering costs in the production of terapixel image. Because each tile contains different information from each other. These complexities affect the processing time and resources required to create the tile. To perform this analysis, we will show the **average gpu usage, power usage, gpu memory usage and elapsed time** required for each pixel on a heatmap.

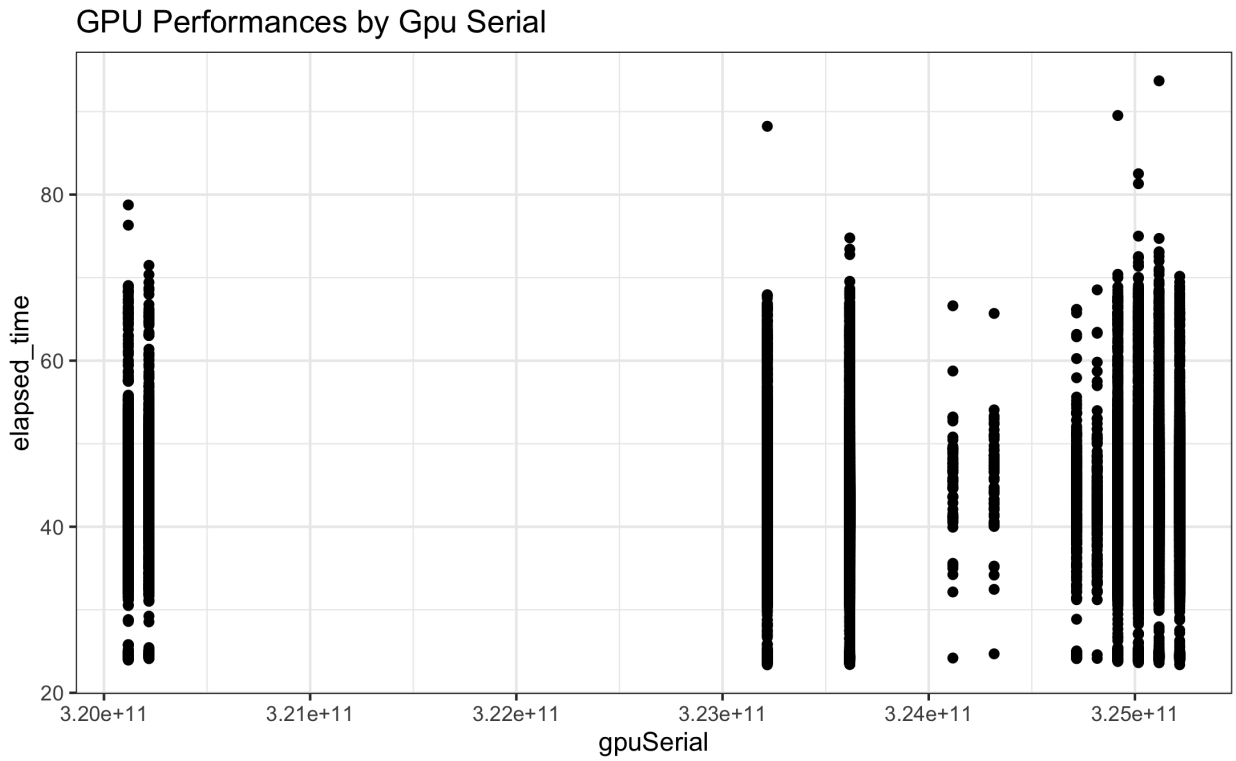
The following heatmaps show the relative impact of rendering each pixel in terms of a given average GPU metrics, time and power. Dark colors indicate low values of the metric, and light colors indicate large values of the metric.

It is clear that the visual legend in the upper left corner of the terapixel image was rendered in a short time using less power. Likewise, the lake at Leazes Park and the structures on the university campus were built with low power and processed quickly. Moreover, buildings and roads are distinguished on the heatmap,because there is difference between the computations required to render streets and buildings. In addition, the initial values of the x-axis seem to be quite intense in the gpu heatmaps, but the elapsed time is quite high. At the beginning of the process, it can be concluded that the system does not work efficiently.



5.5 Can we identify particular GPU cards whose performance differs to other cards?

In the data understanding step, we inferred that there might be a pattern in the gpu serial numbers. Unfortunately, we do not have any information about the capabilities of GPU cards. According to the plot below, it is estimated that there are 12 GPU models.

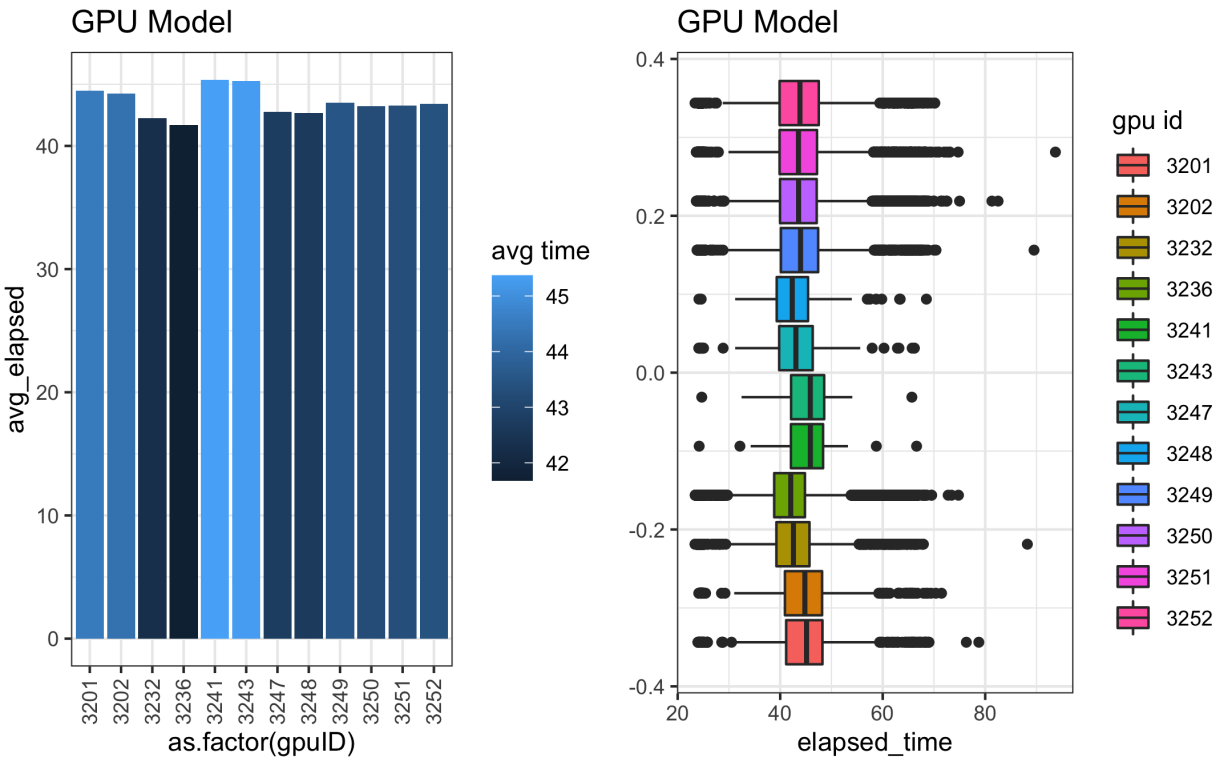


We can divide these serial numbers into 12 according to their first 4 digits.

```
df_elapsed_time_gpu
```

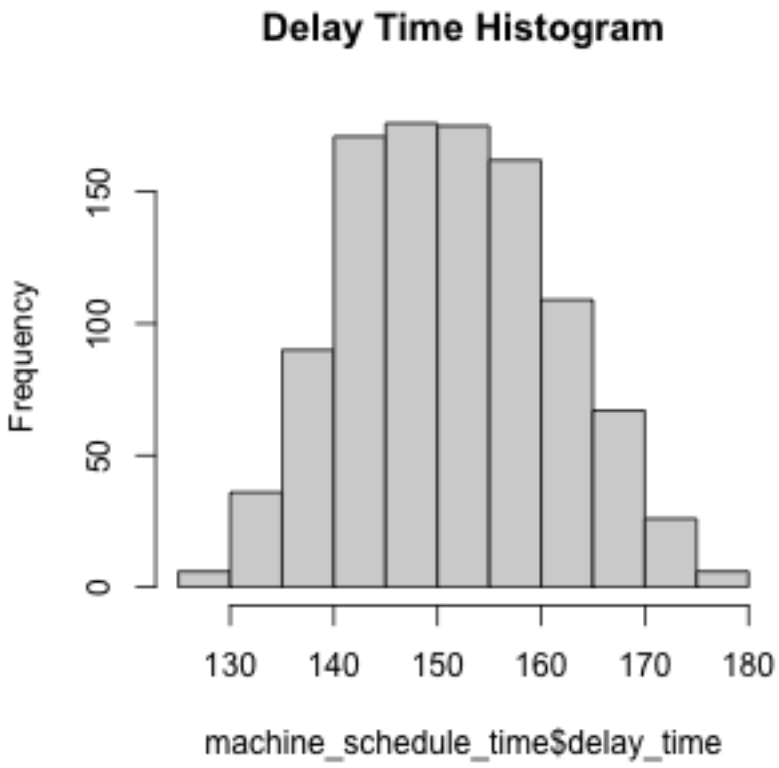
```
## # A tibble: 12 x 3
##        X gpuID avg_elapsed
##    <int> <int>       <dbl>
##  1     1  3201        44.5
##  2     2  3202        44.3
##  3     3  3232        42.3
##  4     4  3236        41.7
##  5     5  3241        45.4
##  6     6  3243        45.3
##  7     7  3247        42.8
##  8     8  3248        42.7
##  9     9  3249        43.5
## 10    10  3250        43.2
## 11    11  3251        43.3
## 12    12  3252        43.4
```

Box chart and bar graph were created to measure the performance of this 12 GPU card model. It was previously stated that the performance metric will be used as the render time. According to the charts below, the performances of the graphics cards are close to each other. However, it is clear that GPU cards with serial numbers beginning with 3236 perform the best, while GPU cards with serial numbers beginning with 3241 and 3243 perform the poorest.



5.6 What can we learn about the efficiency of the task scheduling process?

To measure the efficiency of task scheduling, we need to calculate how much delay there is in scheduling for each machine. We will use the **application.tasks** data set to calculate this. We will create a new data frame, and calculate difference the start of the first render time and the end of the last one for each machine. It will be called the **expected time**. Then, we will sum the total duration of the tasks for each machine. The difference between these 2 calculations will give the machine's total **delay time**.



According to summary, the delay time of machines in the task scheduling process ranges from **125.5 seconds** to **178.8 seconds**. The average time lost is **151.1 seconds**. The 3 machines with the most delays are as follows.

```
df_machine_top_delay_time
```

```
## # A tibble: 3 x 5
##       X expected_time elapsed_time machine_id delay_time
##   <int>         <dbl>        <dbl>      <int>      <dbl>
## 1     1         2912.        2733.        842       179.
## 2     2         2891.        2713.         27       177.
## 3     3         2894.        2717.        719       177.
```

6 Evaluation

The analysis performed for the business goal yielded results on system performance and findings. These results are described and evaluated below.

6.1 Dominated Event Type

Render event accounts for **94.5%** of the time it takes to generate a terapixel image. Uploading 3.19 percent, Tilling 2.23 percent, and Saving config activities 000.5 percent account for the rest. Render activity, which takes an average of **41.2** seconds, dominates the whole process.

6.2 Performance impact on Gpu metrics

Since there was no data on the quality of the tile created, the performance was determined as the time it took to create a tile.

- **Temperature:** After examining the relationship between GPU temperature and performance, it was determined that the standard deviation of temperature had a minor impact on performance.
- **Power Draw:** There is some positive correlation between the median, standard deviation and maximum value of power draw and performance.

6.3 Variation in computation requirements for particular tiles?

The terapixel image's **visual legend** in the top left corner was rendered in a short amount of time and with less power. The rendering time of uncomplicated or monochrome structures and places is also very low, and also there is difference between the computations required to render streets and buildings. In addition, at the beginning of the process, that is, at the left starting points of the the terapixel image, the system does not work efficiently. It takes too much time to render.

6.4 GPU cards with different performance

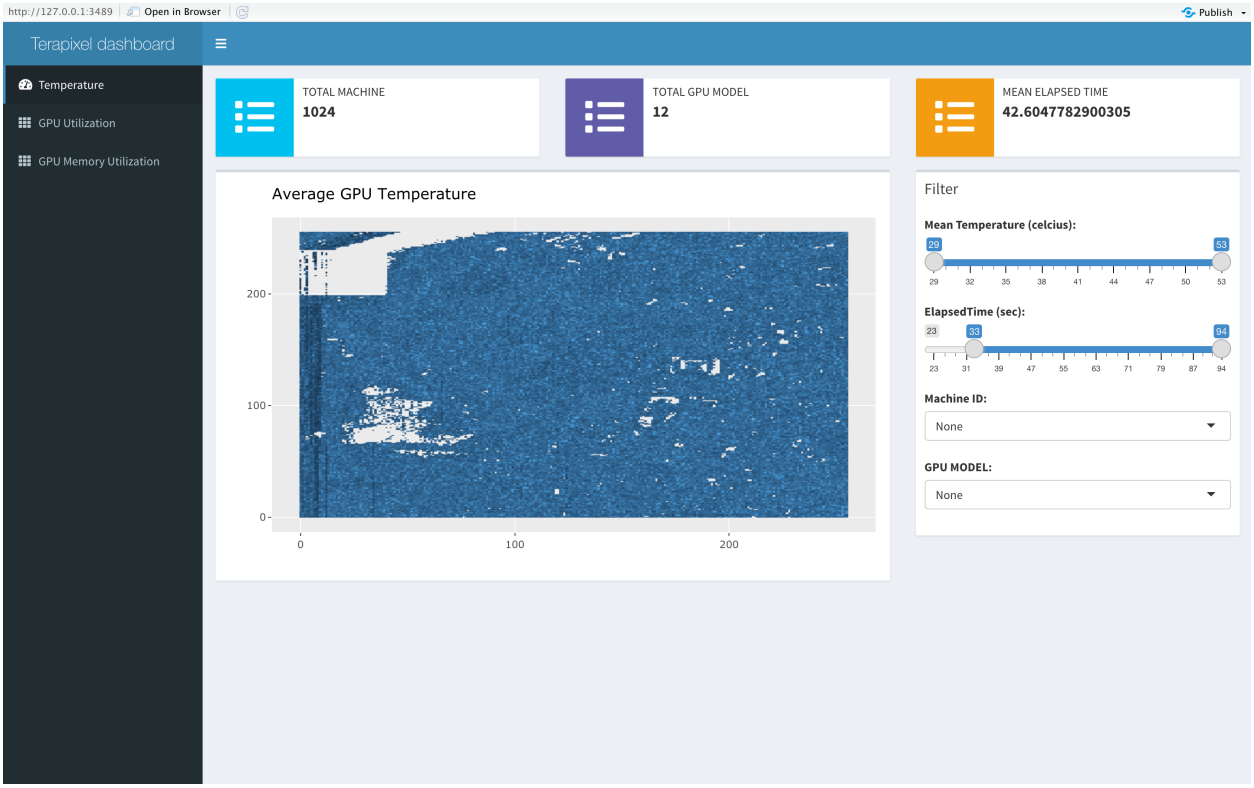
A total of 12 GPU card models were discovered with our approach. The GPU card with the best performance is the one whose serial number starts with **4203** with an average rendering time of 41.6 seconds. On the other hand, the performance of GPU cards starting with **3241** and **3243** is lower than the others. Since there are 65,793 operations in total, the improvement to be made here is critical.

6.5 Efficiency of the task scheduling

We calculated the efficiency of task scheduling based on how much delay there was in scheduling for each machine. As a result, the delay time of machines in the task scheduling process ranges from **125.5 seconds** to **178.8 seconds**. The average time lost is **151.1 seconds**. In other words, each of the 1024 machines in the cloud system does not do a job for an average of 2.5 minutes. The improvement to be made here will increase the performance of the system.

7 Deployment

It doesn't matter how brilliant your findings are, or how precisely your models match the data, if you don't apply them to enhance the way you do business in this last step of the Cross-Industry Standard Process for Data Mining (CRISP-DM) process. As a result, a dashboard was created to view and filter GPU data on tiles. This interface has been developed with the **shiny** dashboard library.



8 Conclusion

The University of Newcastle has implemented a platform project that can provide 3D visualization by collecting data from IoT devices in the city. It carried out in the field of digital twins, visualization is presented with therapist images which containing million megapixels. To produce this type of image, a high-capacity gpu-based cloud infrastructure is used. In this study, we aimed to evaluate the performance and efficiency of the current system and identify improvements for the future. This data mining project has been carried out using the CRISP-DM methodology. Exploratory Data Analysis techniques were used to obtain useful and important information. All analyzes and results are reported in this document.

9 Future Implications

The future implications of this study are that efficiency and performance analysis can be measured by examining the metrics of their systems in large cloud architectures. At the same time, new knowledge may be found through analysis. These studies can increase the efficiency of companies and stakeholders.

10 Personal Reflections

CRISP-DM methodology was applied in this project, providing a uniform framework for planning and managing the project. While developing the project, Git tool was used to speed us up, help us maximize efficiency and provide versioning. Since our data set is a time series, it required high processing power in data manipulations. At this point, writing efficient code was significantly effective. With the Tidyverse package, it helped us a lot in terms of time and performance in data manipulation operations.

11 References

1. Maggie Mae Armstrong. (2020, December 4). Cheat sheet: What is Digital Twin? Internet of Things blog. IBM Business Operations Blog. https://www.ibm.com/blogs/internet-of-things/iot-cheat-sheet-digital-twin/

2. Holliman, N. S., Antony, M., Charlton, J., Dowsland, S., James, P., & Turner, M. (2019). Petascale Cloud Supercomputing for Terapixel Visualization of a Digital Twin. ArXiv.org. https://arxiv.org/abs/1902.04820

3. (2022). Msi.com. https://us.msi.com/page/warranty/vga