Performance Evaluation Of Terapixel Rendering in Cloud(Super) Computing Project

Cahyadi/200511881

1/29/2021

Introduction

Background

Cloud technology, a relatively new technology that is based on utilising the internet infrastructure to the fullest has seen a major increase in popularity in recent years. This success could be attributed to the huge growth of the Tech Industries that seen many new Unicorns; a term first mentioned by Aileen Lee on 2013, used to describe privately owned start-up companies that valued more than 1 billion USD. One of many reasons why the emergence of these "Unicorns" of Tech world brought popularity to Cloud technologies is that Cloud technologies provide a huge benefit to a start-up companies to further expanse and scale their business with greater flexibility and relatively minimum cost and commitment. Start-up companies now have an option to rent the necessary hardware infrastructure instead of investing huge capital on it, since many of the established Tech companies such as Google and Amazon provide access to their extensive range of hardware to be leased. These infrastructures could then be run via virtualisation methods in which provide a great flexibility of scaling. Cloud technologies have been utilised in many parts of the modern society, one of them being for rendering high-resolution images. The implementation of the cloud technologies in rendering task, would eliminate the hardware constraint for the client or the user and expedite the rendering process altogether. An example of this implementation could be found in rendering model in "Image-Based Network Rendering of Large Meshes for Cloud Computing" (Okamoto et al., 2010), in which the model created are able to render any image in real time. A similar cloud implementation are also found in scalable cloud architecture tasked to create a terapixel 3D city visualization of Newcastle Upon Tyne. The architecture uses public IaaS cloud GPU nodes to perform this task.

Typically cloud services provider ascertain that their services are done in regards to their standard, however it is still important that we are able to extract detailed information more suited to our requirements. A result from a journal "Performance Evaluation of Cloud Computing Offerings" (Stantchev, 2009), provide such cases in which there is more information that could be subtracted from the system in which are more relevant for the stakeholders for scalability planning. It is by these reasons that this data mining project is initiated to extract the performance information out of the GPU nodes of the TeraPixel rendering cloud-architecture. The data mining project would focus on the exploratory data analysis, and would be done following CRISP-DM best practice methodology.

Scope

The main aspect of this project is extracting performance information out of an existing cloud-architecture that is meant to create a 3D City visualisation of Newcastle Upon Tyne. The information is extracted by the means of data mining that follows a CRISP-DM best practice steps and would focus entirely on the exploratory data analysis. Based on the results of the analyses done in this project, a conclusion shall be made at the end of the project.

The tools that would be used for this project consist of Rstudio as the main program, supported by Project Template package, and git version control to further streamline the data mining process.

CRISP-DM Methodology

Business Objective

The scalable cloud architecture was created to support the terapixel images rendering in order for it to be accessible for users without putting a constraint on the users hardware. Therefore it might be crucial to be able to extract performance information out of the hardware, in this case, the GPU, to further re-evaluate the amount of GPU nodes and its type needed to run the current rendering process or to create a plan for future usage. The goal of this data mining project is to be able to extract the performance evaluation information out of the current cloud-architecture that is currently used to support the terapixel image rendering. The information extracted would then be able to be used to improve the cost efficiency of running the current cloud-architecture or create future planning to prepare for an expansion. To aid in evaluating the performance of the cloud-architecture rendering process, these questions are provided as a guide for the analyses. The questions are

- Which event dominate the task run time
- Are there any implication that GPU temperature affect the performance of rendering process
- Are there any correlation between the increased power draw and render time
  - Quantify the variation in computation requirements for particular tile
- Are there any particular GPU cards (based on their serial number) that is distinct from the other in terms of performance
- Are there any inefficiencies detected for the task scheduling process

Data understanding

There are three data frames available in which the data mining process could be implemented. These data frames are extracted from the application checkpoint and system metric output from the production of terapixel image. Each data frames are in the comma-separated value format in which it could be loaded directly into R by placing the data frames into the data folder of the Project Template, and loading the project. The three data frames mentioned consist of **Application Checkpoint** data, **GPU** data, and **Task-X-Y** data. Before any process could be made, an initial data exploration would be done on these data frames.

Application Checkpoints Data

Data Preview

```
## # A tibble: 6 x 6
##   timestamp     hostname        eventName  eventType jobId          taskId
##   <chr>         <chr>           <chr>      <chr>     <chr>          <chr>
## 1 2018-11-08T0~ 0d56a730076643~ Tiling     STOP      1024-lvl12-7e~ b47f0263-ba~
## 2 2018-11-08T0~ 0d56a730076643~ Saving Co~ START     1024-lvl12-7e~ 20fb9fcf-a9~
## 3 2018-11-08T0~ 0d56a730076643~ Saving Co~ STOP      1024-lvl12-7e~ 20fb9fcf-a9~
## 4 2018-11-08T0~ 0d56a730076643~ Render     START     1024-lvl12-7e~ 20fb9fcf-a9~
## 5 2018-11-08T0~ 0d56a730076643~ TotalRend~ STOP      1024-lvl12-7e~ 20fb9fcf-a9~
## 6 2018-11-08T0~ 0d56a730076643~ Render     STOP      1024-lvl12-7e~ 3dd4840c-47~
```

Data Summary

```
##   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
```

The Application Checkpoints Data consist of 660400 rows and 6 columns as could be seen from the first 6 rows of the data and the summary table above. For all the columns that are present on the data frame, all of them are character class, and there are no missing values detected. However, after checking for duplicates, there are 130821 duplicated individual within the data set. Therefore, moving onwards with the analyses, the data would be made sure to carry no duplicates.

Assessing each column of the data, a following interpretations could be extracted.

- ***Timestamp*** = Contains the date and the current time of the event down to the milliseconds. The class of this column is that of a character, therefore for further analysis, some processing might be needed to convert the timestamp into a numerical format.
- ***Hostname*** = Contains the Hostname of the virtual machine. The hostname is auto-assigned by the azure batch system. There are 1024 hostname recorded on the application checkpoints data.
- ***Event Name*** = Contains the name of the events that are happening on the task. There are 5 distinct events recorded on the application checkpoints data. These events are
  - Total Render = The entire task itself
  - Saving Config = Saving the configuration
  - Render = The rendering process of the image tile
  - Tiling = Post-processing of the rendered tile
  - Uploading = Uploading the output of the post-processed image tile to the Azure Blob Storage
- ***Event Type*** = Contains two distinct event Type which are STOP and START. This event type are meant to describe the checkpoint of the event name. Meaning, a data point with event name "Tiling" and event type "START" refer to the start of "Tiling" process of a certain task.
- ***Job ID*** = Contains the Azure batch job. There are three distinct value of the Job ID column, and it refers to the level of zoom (4,8,12) of the rendering process.
- ***Task ID*** = Contains the ID of the Azure batch task. There are 65793 distinct task ID. Each task id would have all of the five event names respectively.

GPU Data

Data Preview

```
## # A tibble: 6 x 8
##   timestamp hostname gpuSerial gpuUUID powerDrawWatt gpuTempC gpuUtilPerc
##   <chr>     <chr>        <dbl> <chr>           <dbl>    <int>       <int>
## 1 2018-11-~ 8b6a0ee~   3.23e11 GPU-1d~         132.        48          92
## 2 2018-11-~ d824187~   3.24e11 GPU-04~         117.        40          92
## 3 2018-11-~ db871cd~   3.23e11 GPU-f4~         122.        45          91
## 4 2018-11-~ b9a1fa7~   3.25e11 GPU-ad~          50.2       38          90
```

```
## 5 2018-11-~ db871cd~   3.23e11 GPU-2d~         142.        41          90
## 6 2018-11-~ 265232c~   3.24e11 GPU-71~         120.        43          88
## # ... with 1 more variable: gpuMemUtilPerc <int>
```

Data Summary

```
##   timestamp           hostname           gpuSerial           gpuUUID
##  Length:1048575     Length:1048575     Min.   :3.201e+11   Length:1048575
##  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.: 45.15   1st Qu.:38.00   1st Qu.:  0.00   1st Qu.: 0.00
##  Median : 96.68   Median :40.00   Median : 89.00   Median :43.00
##  Mean   : 89.32   Mean   :40.07   Mean   : 63.23   Mean   :33.47
##  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
```

Extracting the information from the data preview and the data summary above, the GPU data consist of 1048575 rows and 8 columns. The columns consist of Timestamp, Hostname, GPU serial number, GPU UUID, Power Draw on GPU in Watt, GPU Temperature in Celsius, Percent Utilisation of GPU Cores, and Percent Utilisation of the GPU Memory. For the timestamp, hostname, and GPU UUID, the class of column is listed as characters and for the rest of the columns, they are listed as continuous data. However, upon closer inspection, it could be deducted that GPU Serial should not be regarded as a continuous data since it refers to the serial number of the physical GPU card. It seems that this data set also suffers from duplicated values, and after some thorough look at the data set there are 662118 duplicated values found on the data. Similar to the application checkpoints data, the information regarding the duplicated values within the data should be taken into consideration for further analyses to make sure that these duplicated values are not carried over.

Assessing each columns of the GPU data, a following interpretations could be extracted.

- ***Timestamp*** = Similar to the Application Checkpoints Data,the timestamp column for the GPU data contains the current time in which the condition is being recorded. It shows the date as well as the time down to the milliseconds. There is an interesting feature that is found on the timestamp for the GPU data. The timestamp shows that for each hostname, the GPU condition of each hostname is recorded every **2 Seconds**.
- ***Hostname*** = Similar to the Application Checkpoints data, contains the hostname of the virtual machine that is auto-assigned by the Azure Batch system. The number of unique hostname recorded on this data is equal to the number of hostname that is recorded for the Application Checkpoints data, which is 1024 unique hostnames. Therefore, this information could prove to be useful for the analyses on the later part of this report.
- ***GPU Serial*** = Contains the serial number of the physical GPU card. Even though on the data summary the GPU serial column is treated as a continuous variable, on reality, this column should be treated as a categorical variable.
- ***GPU UUID*** = Contains the unique system id which is assigned by the Azure system to the GPU Unit. Note that GPU UUID is unique for each hostname since the number of unique GPU UUID is equal to that of the number of hostname at 1024 unique data points.
- ***GPU Power Draw in Watt*** = Contains the current recorded value of the Power Draw of the GPU in Watt. This variable is a continuous variable. It has a minimum recorded value of 22.55 Watt and a maximum recorded value of 197.01 Watt.

- ***GPU Temperature in Celcius*** = Contains the current recorded Temperature of the GPU in Celcius. This variable is a continuous variable in which the minimum recorded value for this variable is 26 degree Celsius and the maximum recorded value of 55 degree Celsius.
- ***GPU Util Perc*** = Contains the current recorded value of the Percent Utilisation of the GPU cores. This variable is a continuous variable ranging from 0% to 100%. It has a minimum recorded value of 0% and a maximum recorded value of 100%.
- ***GPU Memory Util Perc*** = Contains the current recorded value of the Percent Utilisation of the GPU memory. This variable is continuous variable ranging from 0% to 100%. It has a minimum recorded value of 0% and a maximum recorded value of 83%

Task X Y Data

Data Preview

```
## # A tibble: 6 x 5
##   taskId                      jobId                                  x     y level
##   <chr>                       <chr>                              <int> <int> <int>
## 1 00004e77-304c-4fbd-88a1-1~ 1024-lvl12-7e026be3-5fd0-48ee-b7~   116   178    12
## 2 0002afb5-d05e-4da9-bd53-7~ 1024-lvl12-7e026be3-5fd0-48ee-b7~   142   190    12
## 3 0003c380-4db9-49fb-8e1c-6~ 1024-lvl12-7e026be3-5fd0-48ee-b7~   142    86    12
## 4 000993b6-fc88-489d-a4ca-0~ 1024-lvl12-7e026be3-5fd0-48ee-b7~   235    11    12
## 5 000b158b-0ba3-4dca-bf5b-1~ 1024-lvl12-7e026be3-5fd0-48ee-b7~   171    53    12
## 6 000d1def-1478-40d3-a5e3-4~ 1024-lvl12-7e026be3-5fd0-48ee-b7~   179   226    12
```

Data Summary

```
##     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
```

Extracting information out of the Data Preview and the Data Summary above, the Task-X-Y Data consist of 6579 rows and 5 columns. The number of rows is equal to the number of task as shown by the **taskId** column. It has recurring columns from the Application Checkpoints data in taskId and JobId. This information should be taken into consideration when finding the correlation between the Application Checkpoints data and the Task-X-Y data. The Task-X-Y data columns that shows coordinate "X" and "Y" is deemed by the R system as a column of continuous variable, since the values inside the column consist of numerical value. However, note that since it only shows the coordinate of an object, the stats value extracted from these two columns might not be that beneficial for the analyses, although it still could be useful for determining whether a certain coordinates affect the performance of the rendering task. The "level" column is shown as a continuous variable on the data summary even though the "level" column represent zoom level of the rendered visualisation. There are no duplicates detected within this data frame.

Assessing each column of the Task-X-Y data, a following interpretations could be extracted.

- ***Task ID*** = Similar to the Application Checkpoints data, this column contains the ID of the Azure Batch task. Since it shows the same number of unique taskId as that of the Application Checkpoints data with 65793 unique task ID, it should be taken into consideration to link the task id from the Task-X-Y data with the task id in the Application Checkpoints data.
- ***Job ID*** = Similar to the Application Checkpoints data, this column contains Azure batch job Id
- ***X*** = Contains the X coordinate of the rendered image tile.
- ***Y*** = Contains the Y coordinate of the rendered image tile.
- ***Level*** = Contains the zoom levels of the render. The visualisation is created with 12 zoom levels starting with 1 until 12, however, for the data set in this project, only zoom level 4,8, and 12 that are present since the intermediate level are derived in the tiling process. Note that the zoom level is also reflected in the **JobId** column, as **JobId** also mentions the level of zoom in its character.

Data Preparation

Before analyses would be made, the data frames mentioned and described on the Data Understanding section, would need to be processed first. The process includes but not limited to data wrangling, removing duplicates, merging data frames, and extracting new features from the data. In this part of the report, the processes that are applied to the data frames would be discussed and explanations would be given.

Before any data merging could be done, each individual data frame would be processed first. The data processing is done based on the information gained from the assessment of each data frame that is highlighted in the Data Understanding section of this report.

Application Checkpoints Data

Referring to the initial data assessment of the Application Checkpoints data frame, data points duplicates would be removed. Removing the duplicates on the Application Checkpoints would leave us with a data frame which has 657930 data points and removing 130821 duplicates. The reason to remove any duplicates is to avoid having any error during the analyses or producing an analysis that is skewed.

The **"Timestamp"** column of the Application Checkpoints could not be directly used as a method of analyses since the type of values that are present in the column are characters. Therefore we would process the **"Timestamp"** column and extract only the numerical time value out of the **"Timestamp"**. The dates that are recorded on the **"Timestamp"** are all showing the same date which is ***2018-11-08*** with varying times. Therefore, for the analyses, only the time would be extracted and would be converted into the Hour:Min:Sec format.

For the **Event Name** column, the number of data points for each event names seems to be the same, indicating that for all the task that is present in the data, each task includes all possible event names. The Distribution table of the **Event Name** column is shown as below

```
##
##        Render Saving Config        Tiling   TotalRender     Uploading
##        131586        131586        131586        131586        131586
```

Each event name recorded on the data 131586 times, which if divided by the number of **Event Type** of each Event Name which is 2, would equal to the amount of distinct **Task ID**. With these information at hand,we would proceed with wrangling the data, to prepare it for analyses. The wrangling process are done in these manners

1. Ensure that the Application Checkpoints data does not contain any duplicated data points. If there are any, remove the duplicated data points.
2. Extract only the time characters out of the **Timestamp** column and change the format to that of an Hour:Minute:Second format.

3. Transform the **Event Type** row into two separate columns from the two distinct value of **Event Type** column which are ***START*** and ***STOP*** .
4. Fill these new columns **START** and **STOP** with the **Timestamp** for each **Event Name** which has **Event Type** that matched each of these two new columns.
5. Create a new feature called **duration** which would store the duration of each event, which is calculated by subtracting the **STOP** and **START** column. This newly made feature **duration** is what would be used in analyses as a metric that describe the length of an event.

The resulting data frame would be

```
## # A tibble: 6 x 7
##   hostname eventName jobId taskId STOP           START
##   <chr>    <chr>     <chr> <chr>  <Period>       <Period>
## 1 0d56a73~ Tiling    1024~ b47f0~ 7H 41M 55.921S 7H 41M 55.2S
## 2 0d56a73~ Saving C~ 1024~ 20fb9~ 7H 42M 29.845S 7H 42M 29.842S
## 3 0d56a73~ Render    1024~ 20fb9~ 7H 43M 10.965S 7H 42M 29.845S
## 4 0d56a73~ TotalRen~ 1024~ 20fb9~ 7H 43M 13.957S 7H 42M 29.842S
## 5 0d56a73~ Render    1024~ 3dd48~ 7H 43M 56.239S 7H 43M 16.506S
## 6 0d56a73~ Uploading 1024~ 3dd48~ 7H 43M 57.245S 7H 43M 56.239S
## # ... with 1 more variable: duration <Duration>
```

To aid with the analyses that concerns with the usage of duration column, a horizontal data frame is created in which there are 5 new columns created by using the **Event Name** values as column name and the **duration** of the corresponding **Event Name** would be the values resided within these newly made columns. The resulting data frame would be

```
## # A tibble: 6 x 8
##   hostname jobId taskId Tiling             `Saving Config`      Render
##   <chr>    <chr> <chr>  <Duration>         <Duration>           <Durat>
## 1 0d56a73~ 1024~ b47f0~ 0.720999999999997s 0.00199999999999889s 23.501s
## 2 0d56a73~ 1024~ 20fb9~ 0.99s              0.00300000000000011s 41.12s
## 3 0d56a73~ 1024~ 3dd48~ 0.976999999999997s 0.00199999999999889s 39.733s
## 4 0d56a73~ 1024~ c9e24~ 0.977s             0.00200000000000244s 33.156s
## 5 0d56a73~ 1024~ c5a7a~ 0.911s             0.00199999999999889s 40.571s
## 6 0d56a73~ 1024~ f600a~ 0.927999999999997s 0.00199999999999978s 32.781s
## # ... with 2 more variables: TotalRender <Duration>, Uploading <Duration>
```

Do note that the newly made columns that are based on the **Event Name** column values are not set up in a chronological order in which for each task, the order of event should be, ***Saving Config***, ***Render***, ***Tiling***, and ***Uploading***. We would only use this newly made horizontal data to aid in extracting the average duration of each event name to be used for analyses.

GPU Data

The GPU Data as shown on the Data Understanding part of this report, has some duplicated data points, therefore before any analyses or wrangling is done on this data set, the GPU data set needs to be cleared of this duplicated data points. After removing the data points, the number of rows on the GPU Data would be 1048569 down from the original number of rows for the GPU data of 1048575 data points. Since there is not much information that could be extracted from the GPU data alone, therefore the wrangling that is done for this data set is merely to prepare it for merging this data frame with other data frames on this project. Other than removing any duplicates from the data, the **Timestamp** column of the GPU data would also be transformed into the Hour:Min:Sec format, just as has been done to the **Timestamp** column of the application checkpoints.

Task-X-Y Data

Similar to the GPU data frame, the Task-X-Y data by itself, does not have any noticeable information that could be extracted. And since the Task-X-Y data does not contain any duplicated data points, therefore there is no individual wrangling done to this data.

Merging Application Checkpoints Data with GPU Data

To extract necessary information out of the GPU data, the GPU data needs to be merged with the Application Checkpoints data to gather detailed information regarding the tasks or event tasks that are being run on the GPU nodes at a current point of time. The merged should be by **Hostname** and **Timestamp** columns, however the merge for the **Timestamp** column should not be done so directly, since the **Timestamp** of the GPU data only record the GPU condition every 2 seconds, and if we merged it directly, it would only bear 232 matching results. To tackle this problem, the GPU condition for a particular task is calculated by the average GPU condition during the length of the task itself. An example for this calculation if ***Total Render*** event started at ***8H 1M 29.305S*** and ended at ***8H 0M 44.142S*** we would apply a conditional that would extract all the GPU conditions recorded within this time frame, and for this particular event, we would use the average GPU condition between the two timestamps above to be recorded as the GPU condition of the particular event. However, this conditional too return some problem in the form of **Event Name** which lasted less than 2 seconds. Only using the first conditional would return ***NaN*** value for the GPU conditions for these events. Therefore we would subject another conditional if the average GPU conditions return a ***NaN*** value, we would take the two closest timestamp to **START** and **STOP** column of the event of the same hostname on the GPU data. For example, a ***Saving Config*** event happen between ***7H 41M 45.459S*** and ***7H 41M 45.461S*** and there are no GPU condition that are recorded between those two timestamps for the particular hostname. Therefore, we would take the GPU condition of the same host that is closest to the **START** of the event, which is ***07H 43M 42.443S*** and another which is closest to the **STOP** of the event at ***07H 43M 44.452S*** and we would take the average GPU condition of these two time stamps and recorded it as the GPU condition of the corresponding ***Saving Config*** event.

Data Preview

```
##                                hostname   eventName
## 1 0d56a730076643d585f77e00d2d8521a00000N TotalRender
## 2 0d56a730076643d585f77e00d2d8521a00000N TotalRender
## 3 0d56a730076643d585f77e00d2d8521a00000N TotalRender
## 4 0d56a730076643d585f77e00d2d8521a00000N TotalRender
## 5 0d56a730076643d585f77e00d2d8521a00000N TotalRender
## 6 0d56a730076643d585f77e00d2d8521a00000N TotalRender
##                                            jobId
## 1 1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705
## 2 1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705
## 3 1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705
## 4 1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705
## 5 1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705
## 6 1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705
##                                taskId     eventStart      eventStop duration
## 1 b47f0263-ba1c-48a7-8d29-4bf021b72043 7H 41M 31.687S 7H 41M 56.111S  24.424s
## 2 20fb9fcf-a927-4a4b-a64c-70258b66b42d 7H 42M 29.842S 7H 43M 13.957S  44.115s
## 3 3dd4840c-47f2-4dcc-a775-df2ef6498d71 7H 43M 16.504S 7H 43M 57.245S  40.741s
## 4 c9e249d8-52ed-40c6-8713-b5cbf02ea87e 7H 44M 47.555S 7H 45M 21.898S  34.343s
## 5 c5a7a2df-ddeb-4f54-9cc5-446e3a9ba1ba 7H 45M 24.026S  7H 46M 5.524S  41.498s
## 6 f600a589-3332-4408-a2bd-2b6833981407  7H 46M 7.958S 7H 46M 41.744S  33.786s
##                                   gpuUUID   gpuSerial AvgPowerDrawWatt
```

```
## 1 GPU-1265fef9-aea4-4a5e-8a63-cc5af7b19f4f 3.25217e+11         72.38000
## 2 GPU-1265fef9-aea4-4a5e-8a63-cc5af7b19f4f 3.25217e+11         92.97000
## 3 GPU-1265fef9-aea4-4a5e-8a63-cc5af7b19f4f 3.25217e+11        124.86250
## 4 GPU-1265fef9-aea4-4a5e-8a63-cc5af7b19f4f 3.25217e+11         77.15364
## 5 GPU-1265fef9-aea4-4a5e-8a63-cc5af7b19f4f 3.25217e+11         89.50286
## 6 GPU-1265fef9-aea4-4a5e-8a63-cc5af7b19f4f 3.25217e+11        101.25600
##   AvgGPUTempC AvgGPUUtilPerc AvgGPUMemUtilPerc
## 1    32.33333       69.16667          26.33333
## 2    36.76923       57.46154          31.69231
## 3    39.08333       81.75000          41.83333
## 4    37.90909       71.72727          32.81818
## 5    38.07143       71.00000          33.00000
## 6    38.60000       86.20000          39.60000
```

Merging the Merged Data Frame with Task-X-Y Data

Merging the merged GPU and Application Checkpoints data frame with the Task-X-Y data frame is done by specifying the merge by **Task Id** and **Job Id** columns. For the Horizontal data, the merge would bear 65793 rows while for the vertical data the merge would bear 328965 rows.

Subsetting the data based on Event Name

To streamlined the analyses, the main data would also be subset in terms of its event name and **GPU Serial** column values are extracted as a feature to be recorded into a new column **GPU Serial Feature**. The subset would produce these data frames

- totalRenderData = A data frame that contains only event name ***Total Render***.
- eventNamesData = A data frame that contains all event names apart from ***Total Render*** . In this data frame, the **Event Name** column is also extracted as feature, creating a new feature **Event Name Feature** in which, the event name is turned into a factor, and turned into numerical class for analyses.
- savingConfigData = A data frame that contains only event name ***Saving Config***
- renderData = A data frame that contains only event name ***Render***
- tilingData = A data frame that contains only event name ***Tiling***
- uploadingData = A data frame that contains only event name ***Uploading***

These data subsets are created in order for separate analyses for different event name could be executed.

Data Analyses and Modelling

To reach the goal and answer the questions mentioned in the earlier part of the report, analyses would be made on the data that we have gathered and processed. The analyses would be done in a structure based on the order of the questions mentioned on the Business Objective part of the report. However before any analyses could be made, and initial analysis would be done on the entire data to provide a broad overview of the data and extract some interesting features.

Numerical Summaries of the Data

Extracting the numerical summaries out of the GPU condition and duration of the task would bear us

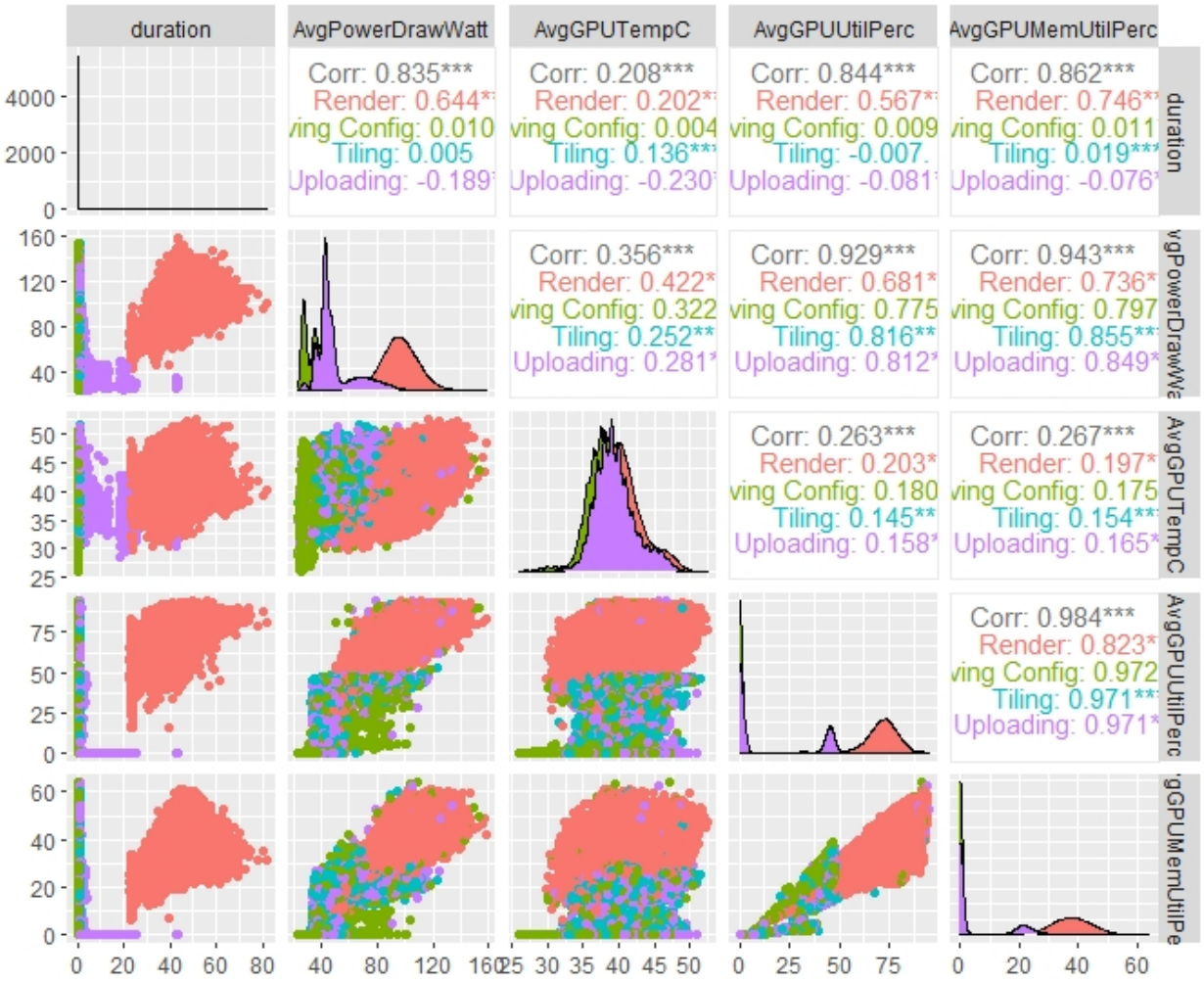
```
##     duration                       AvgPowerDrawWatt  AvgGPUTempC
##  Min.   :0.00199999999999534s      Min.   : 23.45   Min.   :26.00
##  1st Qu.:0.902000000000001s        1st Qu.: 42.50   1st Qu.:37.55
##  Median :1.066s                    Median : 58.40   Median :39.50
##  Mean   :17.2364643685498s         Mean   : 66.28   Mean   :39.70
##  3rd Qu.:40.736s                   3rd Qu.: 91.80   3rd Qu.:41.45
##  Max.   :93.697s (~1.56 minutes)   Max.   :158.60   Max.   :52.43
##  AvgGPUUtilPerc  AvgGPUMemUtilPerc
##  Min.   : 0.00   Min.   : 0.00
##  1st Qu.: 0.00   1st Qu.: 0.00
##  Median :44.00   Median :18.90
##  Mean   :34.18   Mean   :17.58
##  3rd Qu.:68.78   3rd Qu.:34.92
##  Max.   :94.50   Max.   :64.00
```

With variation for each column

```
##          duration  AvgPowerDrawWatt       AvgGPUTempC    AvgGPUUtilPerc
##        423.216000        757.905404          9.393483       1113.057298
## AvgGPUMemUtilPerc
##        320.918051
```

Based on the numerical summaries extracted from the data as shown above, it seems that out of the numerical columns on the data, the column or feature that has the most variation belong to the **Percent Utilisation of GPU Cores**. The **GPU Temperature** seems to have a really low variation. This information might be used to answer the questions stated on the Business Objective of the report in regards to the GPU Temperature.

Graphical Summaries of the Data



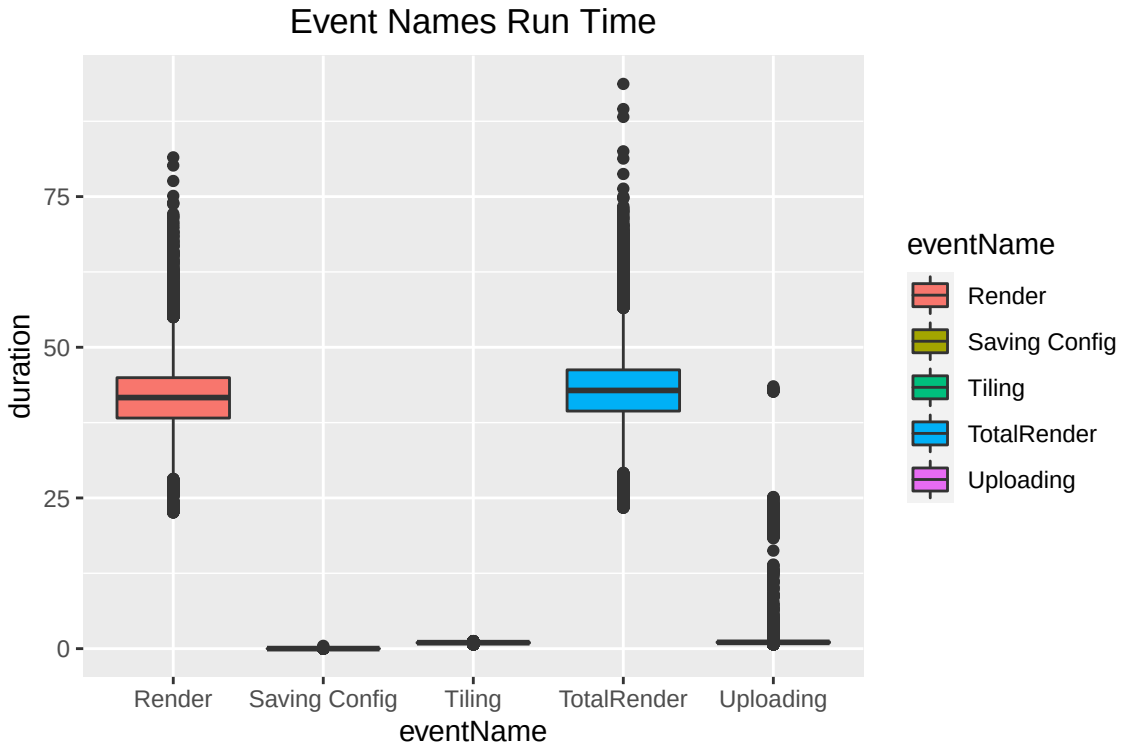
From the graph above, most of the feature that relates to GPU performance such as **duration**, **GPU Power Draw**, **Percent Utilisation of GPU Cores** and **Percent Utilisation of GPU Memory** show that data points that belong to ***Render*** event name tends to reside on the upper side of these graphs. Therefore it might be indicated that ***Render*** demand more performance on the GPU nodes than other event names. It seems there are interesting traits based on the correlation value on some of these combinations of features plots. These interesting correlation traits are found between these variables

- **Duration** and **GPU Power Draw** with a correlation score between the two showing 0.835
- **Duration** and **Percent Utilisation of GPU Cores** with a correlation score between the two showing 0.844
- **Duration** and **Percent Utilisation of GPU Memory** with a correlation score between the two showing 0.862
- **Percent Utilisation of GPU cores** and **GPU Power Draw** with a correlation score between the two showing 0.929
- **Percent Utilisation of GPU memory** and **GPU Power Draw** with a correlation score between the two showing 0.943
- **Percent Utilisation of GPU cores** and **Percent Utilisation of GPU memory** with a correlation score between the two showing 0.984

Apart from the traits shown above, **GPU Temperature** feature doesn't show any strong correlation with any other features shown on the graph above since all the correlation between the **GPU Temperature** and other feature show correlation score below 0.4. This information gathered from plotting the graphs above, would be assessed further to answer the questions stated on the Business Objective part of this report.

Determining Event which Dominate the Task Run Time

For the analysis regarding the question of which event dominates the task run time, we will use the wrangled application checkpoint data with new feature **duration** as a whole. Meaning that the event name still contains the ***Total Render*** event. To answer the question regarding which Event dominates the task run time, a box plot would be plotted to show the spread and mean of each event name duration.



Based on the graph above, a conclusion could be drawn that the ***Render*** event name dominates the task run time since the box plot of ***Render*** event is nearly identical with the ***Total Render*** event name that represent the whole length of the task run time. To quantify the result of the box plot graph above, a numerical summaries are extracted from the data that contains the duration of each event name.

Data Preview

```
## # A tibble: 6 x 8
##   hostname jobId taskId Tiling             `Saving Config`      Render
##   <chr>    <chr> <chr>  <Duration>         <Duration>           <Durat>
## 1 0d56a73~ 1024~ b47f0~ 0.720999999999997s 0.00199999999999889s 23.501s
## 2 0d56a73~ 1024~ 20fb9~ 0.99s              0.00300000000000011s 41.12s
## 3 0d56a73~ 1024~ 3dd48~ 0.976999999999997s 0.00199999999999889s 39.733s
## 4 0d56a73~ 1024~ c9e24~ 0.977s             0.00200000000000244s 33.156s
## 5 0d56a73~ 1024~ c5a7a~ 0.911s             0.00199999999999889s 40.571s
```

```
## 6 0d56a73~ 1024~ f600a~ 0.927999999999997s 0.00199999999999978s 32.781s
## # ... with 2 more variables: TotalRender <Duration>, Uploading <Duration>
```

Extracting the mean out of the data would give us a result of

```
##        Tiling Saving Config        Render   TotalRender     Uploading
##   0.973207210   0.002476266  41.208219552  42.604778289   1.393640524
```

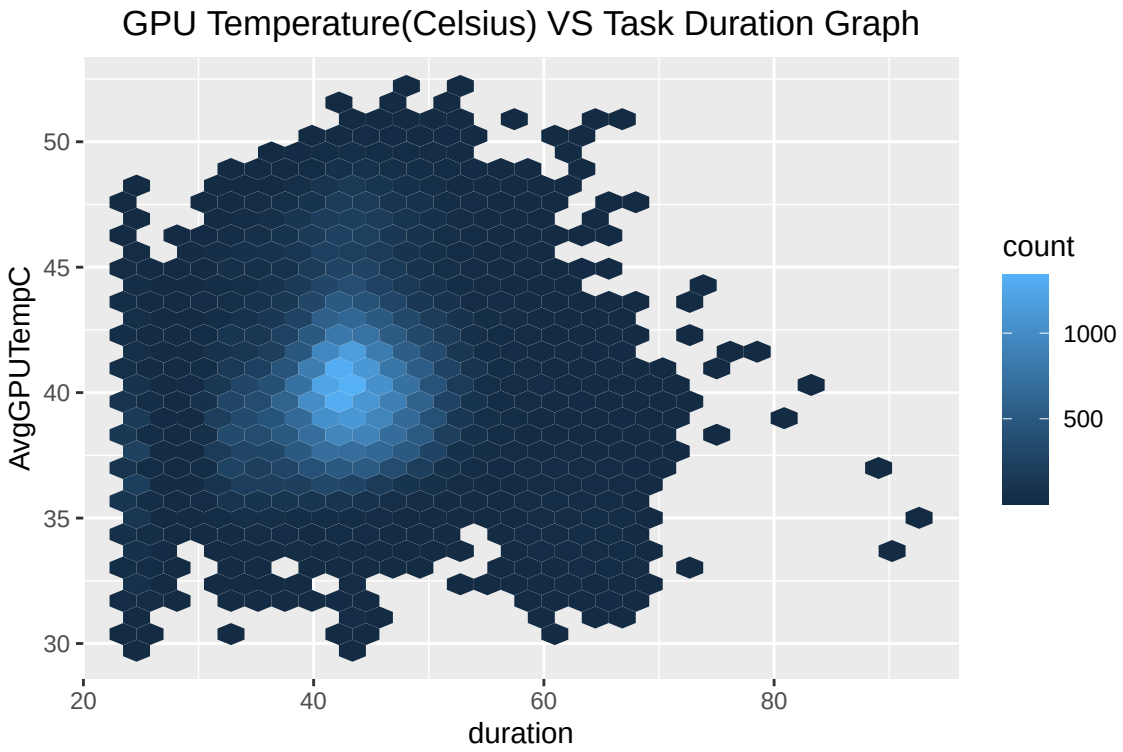
With a variance of

```
##        Tiling Saving Config        Render   TotalRender     Uploading
##  6.951450e-03  7.194054e-06  3.652248e+01  4.222425e+01  6.625830e+00
```

These result further cemented the narrative in which ***Render*** event dominate the task run time by more than 90%. And if we look back at the graphical summaries we achieve using pairs plot produced on earlier part of the analyses they also show that ***Render*** event name resides on the right corner of the graphs which has **duration** feature as its axis, indicating a longer duration time than other event names.

Implications on Temperature of GPU Affecting the Rendering Performance

For the analysis used to answer the question regarding the interplay between the GPU temperature and GPU performance, we would use the **Total Render Data** data frame in which the data contains information of the average GPU condition during the length of an event and only contains event name ***Total Render*** to represent the whole task. GPU capabilities are measured by its performance to successfully render an object. Since there is no information given on the complexity of the object being rendered, it is assumed that each tile of the rendered object are equal in memory size, therefore the metric in which the GPU performance is measured is in terms of the length of the render process. Therefore, to check on the implications on Temperature of GPU affecting the GPU performance, we would like to revisit the plot that shows the correlation between the Duration of rendering task and the temperature of the GPU during this process. To assess this, we would use only the ***Total Render*** event name that represent the whole length of the task itself.



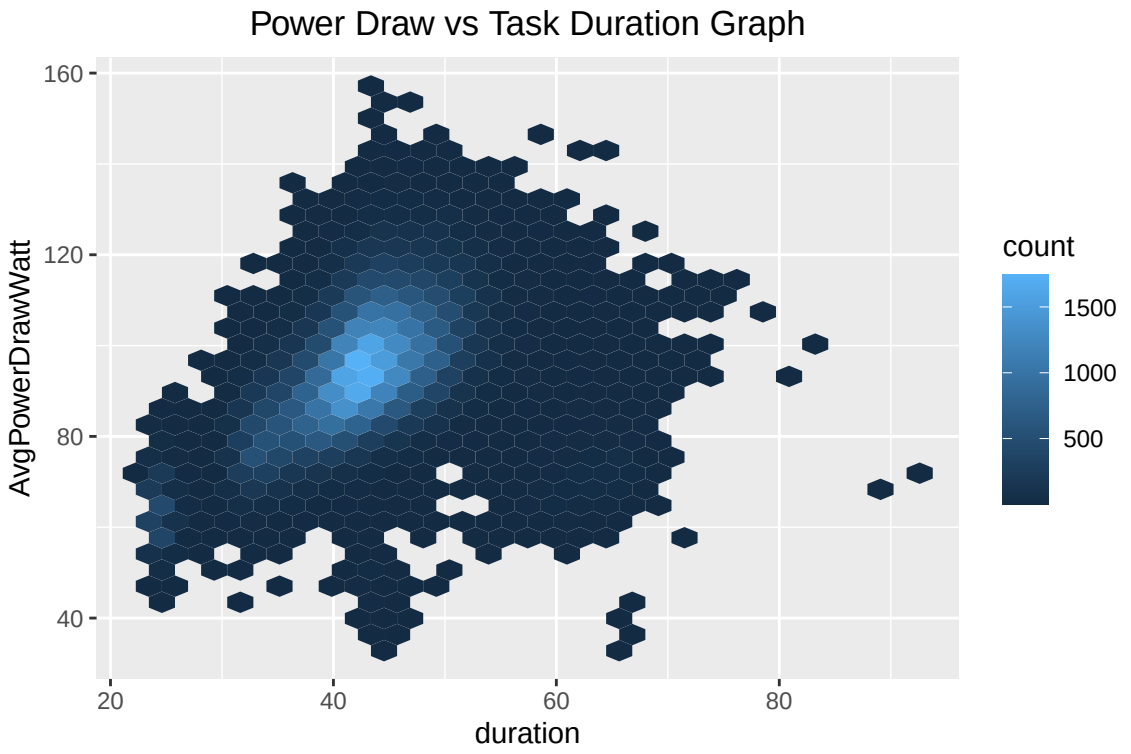
Based on the GPU Temperature vs Task Duration graph,it seems that there are no strong correlation between the two variables. The graph didn't show any noticeable trend and the spread of the data appeared randomly scattered and there seems to be a high volume of data points around the mean of both variable. To further quantify this deduction, the correlation matrix between the **GPU Temperature** and the **Duration** of the task show a score of

	duration	AvgGPUTempC
duration	1.0000000	0.1037189
AvgGPUTempC	0.1037189	1.0000000

The deduction from the GPU Temperature vs Task Duration Graph, and further supported by the correlation matrix between the two variables, shows that there are no implications that Temperature of GPU affect the Performance of the GPU rendering capabilities that are measure by the length of the render as the performance metric.

Connection Between the Increased Power Draw and Render Time and Computational Variation for Particular Tile

For this analysis, we would use the **Total Render Data** data frame once again. To answer the question regarding the correlation between an increased power draw and the duration of render, a graph between the two feature variable **GPU Power Draw** and **duration** would be plotted and produce result of

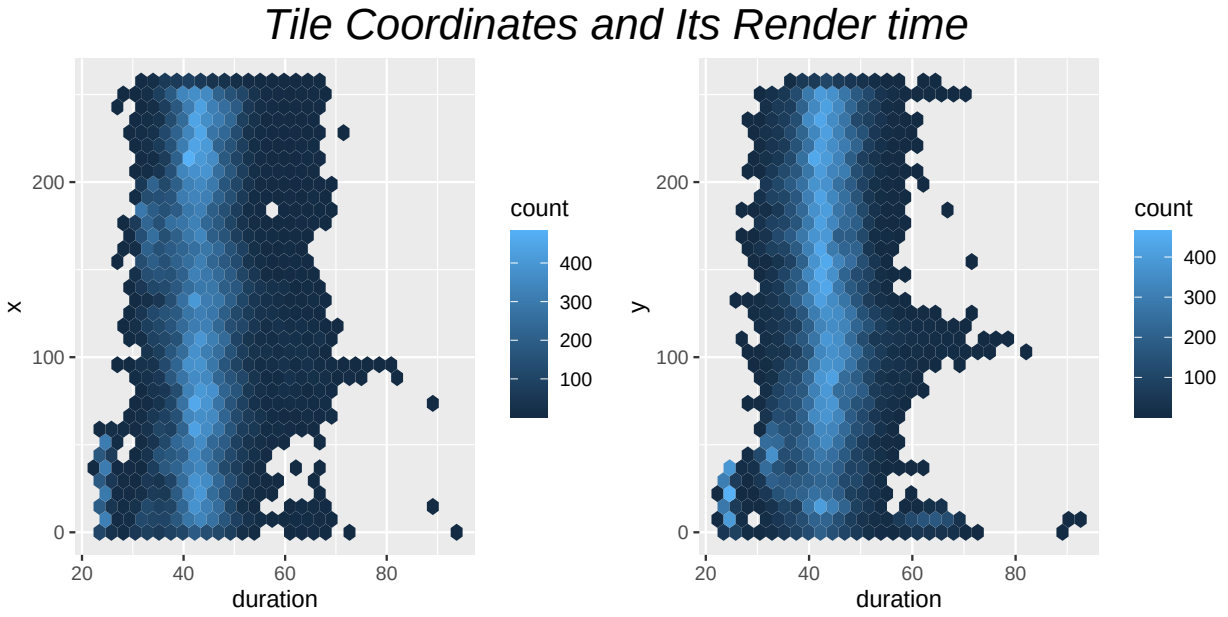


Looking at the graph above, there is a slight positive correlation trend is detected between the two variables. However, based on this graph alone, there is no clear indication that increased power draw resulted in longer duration or the other way around. Therefore to better understand the correlation between the two variables, a correlation matrix is produced.

	duration	AvgPowerDrawWatt
duration	1.0000000	0.5018929
AvgPowerDrawWatt	0.5018929	1.0000000

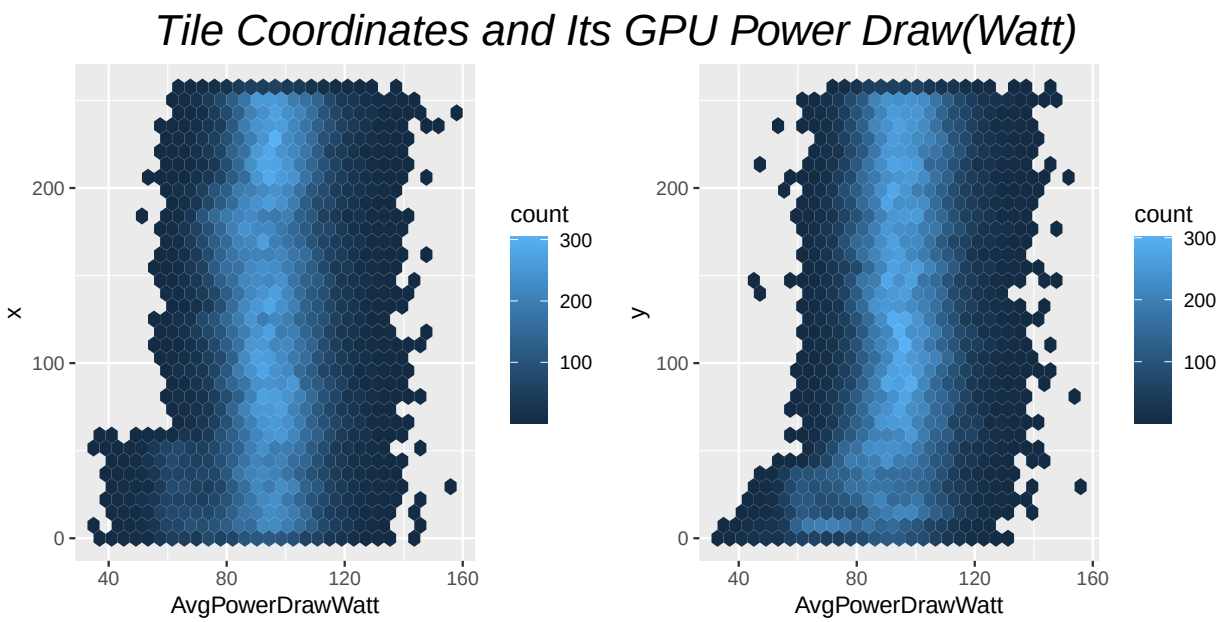
The correlation matrix shows that there is some correlation between the **GPU Power Draw** and the **Duration** of the render task, however, the correlation is not strong at *0.5018929*, therefore it could be deducted that the **Duration** of an event task does not influence the **Power Draw** or the other way around. Therefore, it prompt a question on what truly drives the increase in **Power Draw** or what caused the render task to run longer. In an attempt to answer these, we would like to perform analysis on the image tile itself (X and Y coordinate) in regards to its rendering time and the variation of the computational requirements for a certain tile.

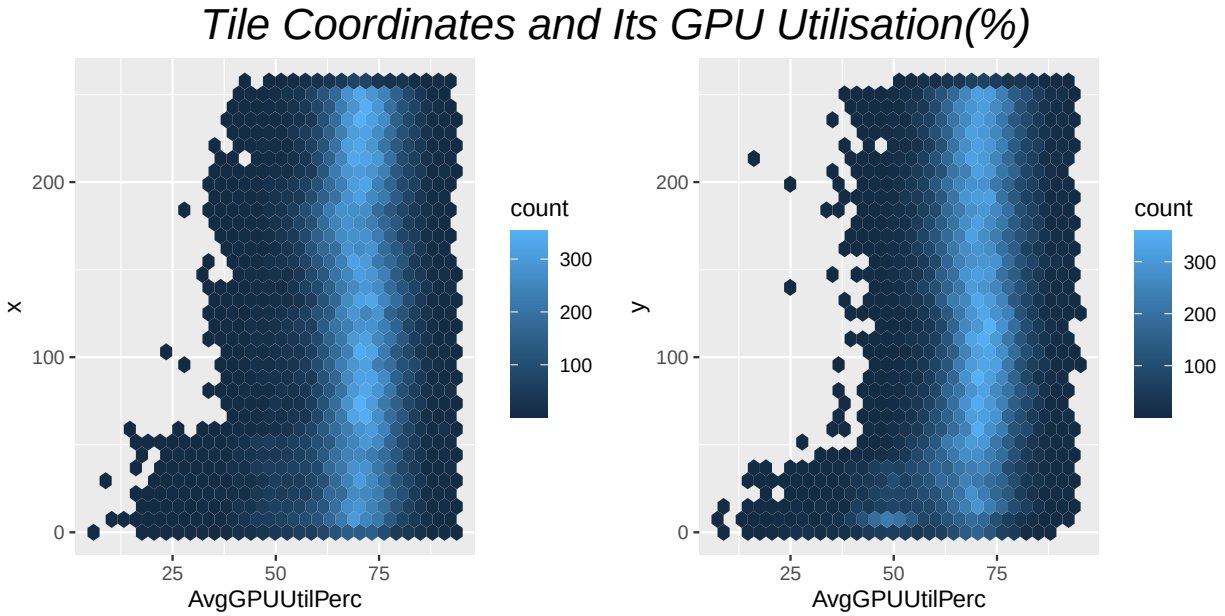
First, we would like to check each coordinate axis and observe whether certain coordinate points are rendered more slowly compared to the average render time of other coordinates within the same axis. Therefore, a hex plot graph is produced

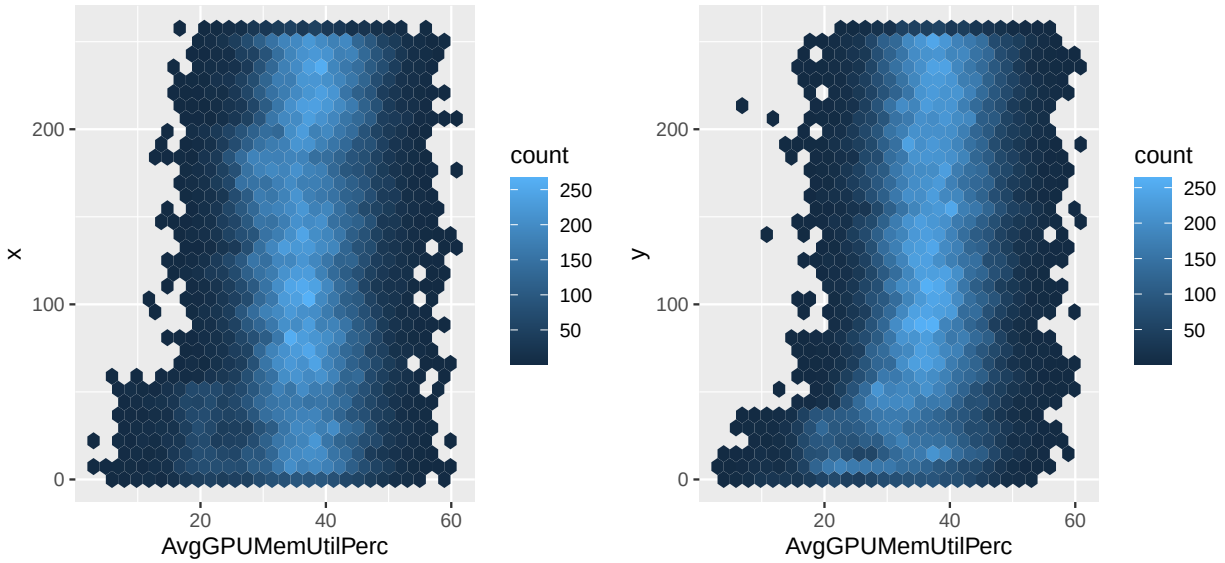


Based on the graphs above, for the X-axis coordinates graph, it seems the tiles that have render time which is more than 70 seconds are X coordinates between 100, and as for the Y-axis coordinates graph, the same could be said the same that there are tile which belongs around the 100 Y-coordinates that took more than 70 seconds to render. However, on the Y-coordinate graph, there are also a small cluster of tiles which took more than 60 seconds to render. The tiles in question have Y-coordinate around 0.

Now to look at computational variance for different tile coordinates we would plot graphs for each computational columns of the data.







Based on the GPU conditions graph in regards to Image tile coordinate above, there is no clear correlation that could be drawn between the position of image tile being rendered and the increase of computational requirement. However, there is one more feature that we could explore, which is the level of render which contributes to the level of zoom of the image.

The **level** feature is part of the **Task-X-Y** in which it is provided with the image tile coordinates. However upon closer observation on the data, only **1** image tile coordinate that is being rendered at Zoom Level 4 and only **256** image tile being rendered on Zoom level 8, while all the image tile coordinate are subjected to render at Zoom level 12. Therefore, only tiles available to all Zoom level would be compared. Below are the comparison of the GPU condition of the tile at (0,0) coordinate at different Zoom level.

duration	AvgPowerDrawWatt	AvgGPUTempC	AvgGPUUtilPerc	AvgGPUMemUtilPerc	level
52.182s	100.17789	36.21053	77.52632	41.84211	4
43.853s	43.70286	31.80952	29.14286	12.00000	8
24.627s	62.37400	39.00000	52.50000	21.80000	12

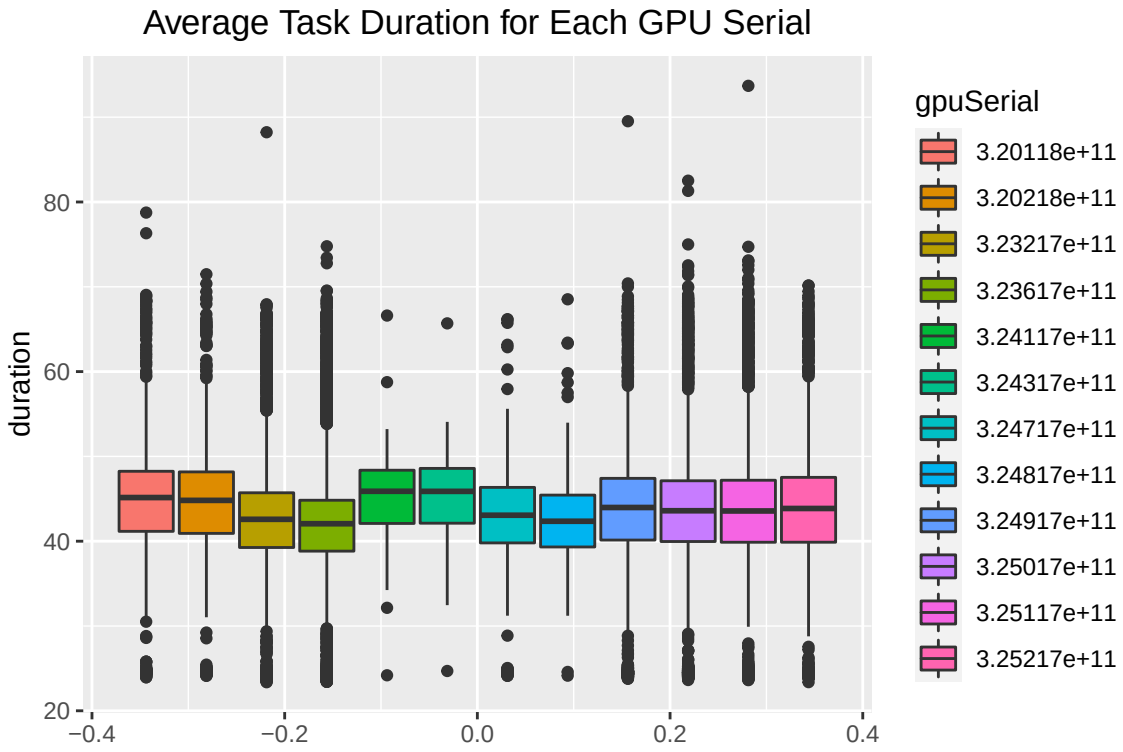
From the table provided above, it seems that increase in zoom level would alleviate the load on the GPU since lower zoom level contributes to a wider area that needs to be covered by the render due to the image being more zoomed out. We would check the comparison of average GPU condition between the 256 image tile that are being rendered on both level 8 and level 12.

duration	AvgPowerDrawWatt	AvgGPUTempC	AvgGPUUtilPerc	AvgGPUMemUtilPerc	level
48.45982	87.59534	37.40291	63.72880	32.37927	8
27.22477	61.76007	35.89550	48.15191	19.12198	12

It is further confirmed by the table of comparison between Zoom Level 8 and Zoom Level 12 that the more zoomed out an image is, the more it demands higher performance out of the hardware.

Identifying GPU Card whose Performance Differ from Others

There are a total of 12 types of physical GPU cards used on the current Cloud Architecture, and we would like to gather the information regarding the performance of each type of GPU cards. To aid in identifying which GPU card perform better or worse than the rest, we would plot a graph in which it will show the average render run time for each type of GPU cards.



From the graph, we could observe that out of the 12 GPU cards type, there are 2 that performs slightly worse than the other which are GPU with serial number of ***3.24117e+11*** and ***3.24317e+11*** . To quantify the graph, we extract the average run time for each type of GPU cards.

gpuSerial	duration
3.20118e+11	44.50196
3.20218e+11	44.26358
3.23217e+11	42.26390
3.23617e+11	41.68215
3.24117e+11	45.36851
3.24317e+11	45.29177
3.24717e+11	42.75674
3.24817e+11	42.67423
3.24917e+11	43.50489
3.25017e+11	43.24727
3.25117e+11	43.29552
3.25217e+11	43.42346

From both the graph and the numerical summaries above, we could deduct that out of the 12 distinct GPU Serials, the two GPU Serials that perform the worst are GPU Serial ***3.24117e+11*** and ***3.24317e+11*** with average render run time of ***45.36851 Seconds*** and ***45.29177 Seconds*** respectively. And the GPU Serial that perform the best is GPU Serial ***3.23617e+11*** with average render run time of ***41.68215 Seconds***. The difference might not be much at glance but if we take into account the number of tiles that needs to be rendered which is 655536 tiles in total even after considering all 1024 hosts working together at the same time, even 1 second difference in average rendering time would result in great inefficiency. Therefore it should be taken into consideration to opt for the fastest type of GPU cards.

Efficiency of the Task Scheduling

For this analysis, we would use the **Total Render Data** data frame. To extract information regarding the efficiency of task scheduling, we would focus on the **Timestamp** column of the data frame. The data that we would work on is the subset of the **Durations** data frame in which we only interested on the ***START*** timestamp of the task and the ***STOP*** timestamp of the task. The inefficiency would be measured by the amount of idle time between tasks. For this analysis we would assess each hostname separately and we would record the total idle time for each hostname. The result of this analysis would be in the form of Inefficiency data found below.

Data Preview

Hostname	TimeLoss	RecordedLength
0d56a730076643d585f77e00d2d8521a00000N	173.676s (~2.89 minutes)	1H -12M -8.755S
5903af3699134795af7eafc605ae5fc700000F	145.968s (~2.43 minutes)	1H -12M 20.234S
5903af3699134795af7eafc605ae5fc7000003	141.792s (~2.36 minutes)	1H -12M 13.568S
265232c5f6814768aeefa66a7bec6ff6000008	141.895s (~2.36 minutes)	1H -12M 20.772S
a77ef58b13ad4c01b769dac8409af3f8000017	150.601s (~2.51 minutes)	1H -12M 23.055S
a77ef58b13ad4c01b769dac8409af3f8000005	171.451s (~2.86 minutes)	1H -12M 24.124S

From the table of idle time for each hostname or GPU nodes above, it seems that there is an average of ***151.2255059 Seconds*** idle time out of ***1 hour and 12 mins*** average length of time of the GPU condition being recorded. Even though the idle time seems small, if this idle time is times by the number of hours

within a day and times by 365 days within a year, this inefficiency in terms of idle time would mean extra loss of money. Therefore the task scheduling efficiency should be improved. One of the ways to improve the task scheduling efficiency is to schedule the next rendering task to be done to be run right after the first task has just finished the process.

Evaluation

The analyses that have been done above, are meant to answer the questions mentioned on the Business Objective Part of the report. Based, on the results of the analyses, each question has been able to be answered and following are the foundings from the analyses that should be considered in creating future scalability planning for the cloud-architecture for rendering the 3D city Visualisation of Newcastle upon Tyne.

- The task run time is dominated by the ***Render*** event
- There is no implication that there is an interplay between **GPU Temperature** and the **Duration** of the task
- There is a slight correlation between the increased power Draw and the increase of Render time
- Lower zoom level results in higher computational requirement.
- 2 GPU Card types are slower than the rest which are GPU cards of serial number ***3.24117e+11*** and ***3.24317e+11***
- And it seems that the current task scheduling is inefficient in that there are gaps between when the task is finished and another task is started.

In regards to the foundings above, a following recommendation could be made.

- Opting to switch to the faster type of GPU cards and avoid using GPU cards with serial number of ***3.24117e+11*** and ***3.24317e+11*** if possible.
- Improve the task scheduling efficiency by programming tasks to run right after prior task has finished.

Conclusion

Based on the results found from data mining using the CRISP-DM methodology, the project could be seen as a moderately successful project. Since this project is mainly an Exploratory Data Analysis on the performance data of the GPU nodes used to produce a terapixel image, therefore the metric in which the success of the project is determined is the ability of the data mining process to extract useful information, and the data mining process in this project are able to fulfil that requirement by successfully answering all the questions mentioned on the earlier part of this project. The advantage of using this evaluation metric to measure the success of an EDA data mining project is that it was able to bottleneck the goal and objective of the data mining process to answer the questions stated on the earlier part of the project. The downside, however, that it focuses only to answer the questions mentioned in the earlier part of the project that it might leave out other interesting traits that are not related to the questions.

Future Works

Future implications on the work of this area are that with the rapid growth of the current cloud technology, the number of ways to extract performance information out of the cloud - architecture is increasing. The implications are supported by the number of work regarding performance evaluation in the fields, such as the works by Stantchev and Papdopoulos et al. The increased in performance evaluation works is also driven by the needs of companies and stakeholders to further improve their efficiency and grow their business further.

Possible extension of this project is a scalability planning that could use the information gain from this project as a consideration.

Personal Reflections

Undertaking this project has increased my familiarity of using CRISP-DM best data mining practice methodology,and using the CRISP-DM methodology along with Project Template package has truly helped in streamlining the whole data mining process. Throughout the project, the most difficult process to be executed perfectly is the data wrangling process in which we need to redo the wrangling process until an appropriate data frame is produced. The amount of data that is being wrangled is also a driving factor that caused the lengthy wrangling process. Apart from the wrangling process, the rest of the processes done in this project are quite straight forward as a result of using the CRISP-DM methodology. The usage of Project Template has also helped in streamlining the data logistic processes. Last but not least, the usage of Git version control has also helped in creating a framework that enables us to have the flexibility to go back to a certain "commit" version of the program. Therefore the CRISP-DM methodology combined with the Project Template package of R and Git version control, would be used for similar project going forward.

Reference

- Okamoto, Y., Oishi, T. & Ikeuchi, K. Image-Based Network Rendering of Large Meshes for Cloud Computing. Int J Comput Vis 94, 12–22 (2011). https://doi.org/10.1007/s11263-010-0383-1
- V. Stantchev, "Performance Evaluation of Cloud Computing Offerings," 2009 Third International Conference on Advanced Engineering Computing and Applications in Sciences, Sliema, 2009, pp. 187-192, doi: 10.1109/ADVCOMP.2009.36.
- A. V. Papadopoulos et al., "Methodological Principles for Reproducible Performance Evaluation in Cloud Computing," in IEEE Transactions on Software Engineering, doi: 10.1109/TSE.2019.2927908.