PERFORMANCE EVALUATION OF TERAPIXEL RENDERING IN CLOUD (SUPER) COMPUTING

Cloud Computing Project

SATISH PILLA

B210472095

Contents

1 Introduction

We live in an era where enormous datasets are continuously being used to fuel scientific discoveries. Scientists today envision a variety of data analytics and computations ranging from desktop to supercomputers, but they often struggle to design and create software architectures that can manage heterogeneous and often inconsistent data on a large scale.

Furthermore, the demand for scientific data and computational resources can fluctuate dramatically over time. Analytical demand may have broad transients in reaction to seasonal field campaigns or new breakthroughs in instrumentation [1]. Cloud computing can provide a flexible, cost-effective, and well-suited platform for some of these growing science needs. One such solution was used to present the terapixel visualization of IoT sensed environment data about the city of Newcastle, the data was collected by Newcastle Urban Observatory. Newcastle university team provided a solution which is a scalable architecture for a cloud-based visualization that can be deployed and pay for only as needed. They were able to demonstrate that it is feasible to produce a high quality terapixel visualization using a path tracing renderer in under a day using IaaS cloud and GPU nodes.

Terapixel Image:

Terapixel images are images with over one trillion pixels that, when combined with the correct toolset, give an intuitive [2], fluid user experience that allows the viewer to see an overview of the entire image or zoom into great detail.

Rendering:

Rendering, also known as image synthesis, is the automatic process of creating a photorealistic or non-photorealistic image from a 2D or 3D model (or models in what is referred to as a scene file) using computer programs. A render can also be defined as the outcome of presenting such a model.

2 Business Understanding

2.1 Background

Terapixel images provide stakeholders with an intuitive and accessible way to show data sets, allowing audiences to interactively search huge data at many scales. The Newcastle Urban Observatory has been collecting IoT-sensed environmental data about Newcastle-upon-Tyne for over many years, accumulating more than 900 million records of data to date. The issue we covered here is how to offer the resources needed on a supercomputer scale to measure a realistic simulation of Newcastle upon Tyne's city terapixel image and environmental data recorded by the Newcastle Urban Observatory.

Newcastle university provided a solution which is a scalable cloud-based visualization platform that we can deployed and pay for only when it is needed [3]. The three main goals of a study of IoT-sensed environment data are to develop a supercomputer architecture for scalable visualization in the cloud, to create a terapixel 3D city visualization that can be updated on a daily basis, and to conduct a comprehensive evaluation of cloud supercomputing for compute-intensive visualization applications. They showed that combining public IaaS cloud GPU nodes and a path tracing renderer, can create a high-quality terapixel visualization in under a day. Once the terapixel image has been created, it may be used to support the city's interactive browsing and data at a variety of sensing scales, from the entire city to a single desk in a home, and it can be accessed from anywhere in the world.

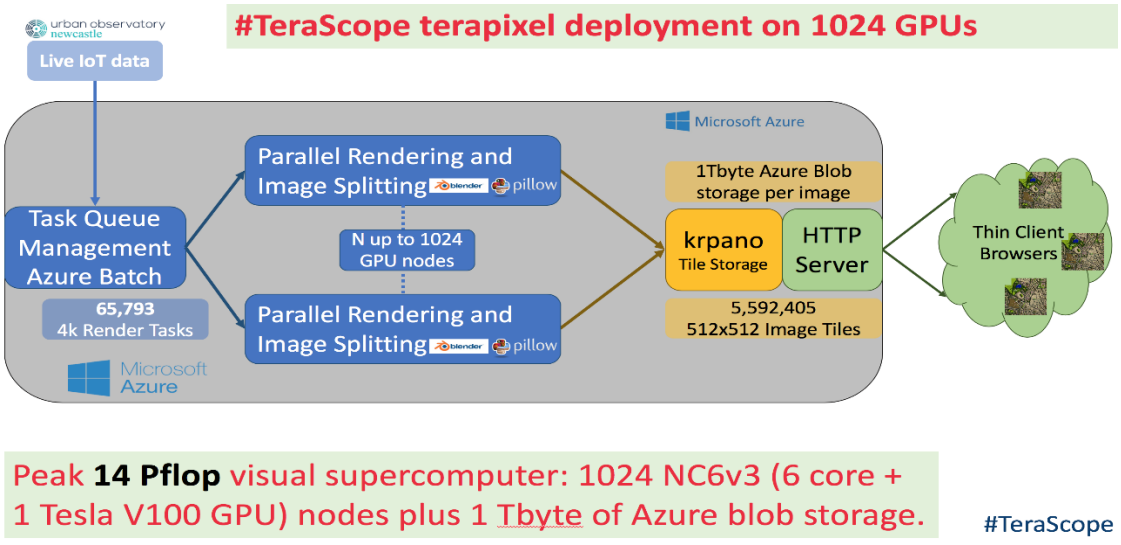


Fig 1: Block Diagram of Terapixel Rendering

The main components of terapixel rendering are Urban Observatory data which collected the data, Microsoft Azure (to deploy the solution), client. The data collected by Urban Observatory is from the live IoT devices. The data is then sent to Azure in which the task queue management Azure batch assign the data to different jobs created on the virtual machine for rendering and splitting the image. Various events are performed in each job with a unique task id for each rendering. There will be GPU nodes that collect output metrics related to the virtual machine's GPU status. The tiles are then saved in krpano after rendering, which consumes 1Tbyte per image. An http server delivers these images to visitors using client browsers.

Fig 2: visualization of the realistic terapixel image of Newcastle upon Tyne city

2.2 Objective

The purpose of this project is to look into bottlenecks and trends in the use of GPUs in Newcastle Upon Tyne's terapixel rendering operations on a cloud computing platform. The following are the expected criteria to investigate.

- Which event types dominate task runtimes?
- What is the interplay between GPU temperature and performance?
- What is the interplay between increased power draw and render time?

2.3 Tools & Methodologies

We are following CRISP-DM methodology for this project, In the project preparing the data before exploration and understand. There will be no modelling and deployment steps in the project.

Tools:
Python Programing language, for reproducibility we are using juypter notebook.
Git and Github is used as version control system.

3 Description of Data

The dataset was collected from application checkpoint and system metric output from the production of a terapixel image during a run using 1024 GPU nodes. The data shows the performance of the GPU card, performance timing of the render application and the details of each image during rendering.

The dataset in our projects:

- Application-checkpoints.csv
- gpu.csv
- task-x-y.csv

Application-checkpoints file: This file contains application checkpoint events throughout the execution of the render job. The different schema present in the application-checkpoint data file is

timestamp, hostname, eventName, eventType, jobId and taskId.

- **Hostname:** This is the host name of the virtual machine which is auto-assigned by the Azure batch system.
- **eventName:** This column consists of 5 events occurring in the application during rendering. The 5 events are:
    - 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
- **eventType:** It has two possible values START and STOP.
- **jobId:** This is the jobId of the Azure Job. The run is split into 3 jobs.
- **taskId:** This consists of unique taskId's for each render task.

gpu.csv file: The data file contains metrics which have been generated on the virtual machine regarding the status of the GPU. The schema presents in the gpu.csv dataset is:

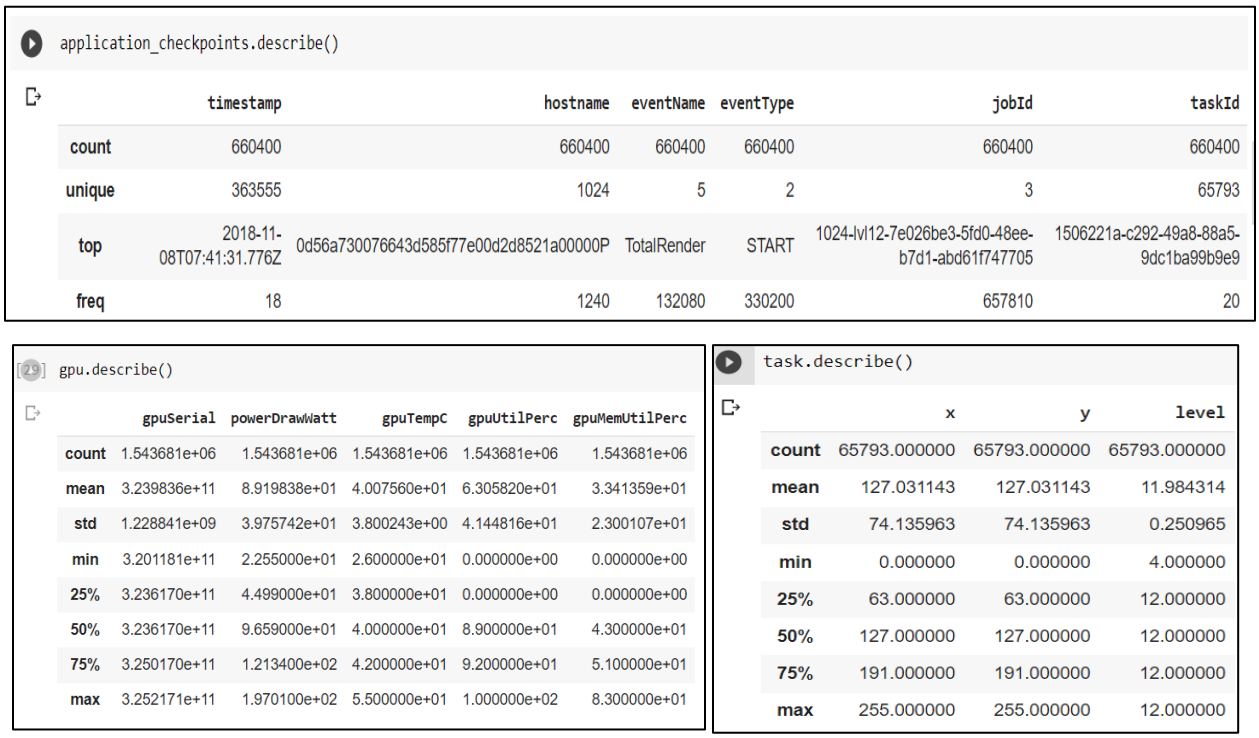
- **gpuSerial:** It consists of the serial number of each physical GPU card.
- **gpuUUID:** The unique device ID is allocated to the GPU unit by the Azure systems.
- **powerDrawWatt:** It consists of the power draw of the GPU units in watts.
- **gpuTempc:** this has the temperature of the GPU which is noted in Celsius.
- **gpuUtilPerc:** it consists of the percentage utilization of the GPU cores.
- **gpuMemUtilPerc:** it consists of the percentage utilization of the GPU memory.

Task-x-y.csv file: This data file contains the x, y coordinates for each task, from which part of the image was rendered. The schema in this file is:

- **jobId:** Id of the Azure batch job.
- **taskId:** ID of the Azure batch task.

- **x:** X co-ordinate of the image tile being rendered.
- **y:** Y co-ordinate of the image tile being rendered.
- **Level:** The visualization created is a zoomable "google maps style" map. In total we create 12 levels. Level 1 is zoomed right out and level 12 is zoomed right in. You will only see levels 4, 8 and 12 in the data as the intermediate level are derived in the tiling process.

Descriptive stats of all the three datasets are as follows:



application_checkpoints.describe()

	timestamp	hostname	eventName	eventType	jobId	taskId
count	660400	660400	660400	660400	660400	660400
unique	363555	1024	5	2	3	65793
top	2018-11-08T07:41:31.776Z	0d56a730076643d585f77e00d2d8521a00000P	TotalRender	START	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	1506221a-c292-49a8-88a5-9dc1ba99b9e9
freq	18	1240	132080	330200	657810	20

gpu.describe()

	gpuSerial	powerDrawWatt	gpuTempC	gpuUtilPerc	gpuMemUtilPerc
count	1.543681e+06	1.543681e+06	1.543681e+06	1.543681e+06	1.543681e+06
mean	3.239836e+11	8.919838e+01	4.007560e+01	6.305820e+01	3.341359e+01
std	1.228841e+09	3.975742e+01	3.800243e+00	4.144816e+01	2.300107e+01
min	3.201181e+11	2.255000e+01	2.600000e+01	0.000000e+00	0.000000e+00
25%	3.236170e+11	4.499000e+01	3.800000e+01	0.000000e+00	0.000000e+00
50%	3.236170e+11	9.659000e+01	4.000000e+01	8.900000e+01	4.300000e+01
75%	3.250170e+11	1.213400e+02	4.200000e+01	9.200000e+01	5.100000e+01
max	3.252171e+11	1.970100e+02	5.500000e+01	1.000000e+02	8.300000e+01

task.describe()

	x	y	level
count	65793.000000	65793.000000	65793.000000
mean	127.031143	127.031143	11.984314
std	74.135963	74.135963	0.250965
min	0.000000	0.000000	4.000000
25%	63.000000	63.000000	12.000000
50%	127.000000	127.000000	12.000000
75%	191.000000	191.000000	12.000000
max	255.000000	255.000000	12.000000

Fig 3: Descriptive stats for application checkpoint, gpu, task dataset

From above we can see the application dataset contains over 6 lakh records, gpu dataset contains over 15 lakh records and task dataset contains over 65 thousand records.

4 Data Preprocessing

In our project we are given three files, application, gpu, task file. At first, we are converting timestamp for application and gpu in date time format. Secondly, we are merging application and task file based on taskId and jobId and named the dataset as df_app_task_merge, then we are sorting the merge file based on eventType start and stop for future analysis. At the end we are merging df_app_task_merge and GPU file based on timesatamp to get our merge_df dataset (combination of all three files). Initially while exploring we will first try to explore individual dataset and later, we will try to explore merge dataset.

5 Exploratory Data Analysis

Data scientists utilize exploratory data analysis (EDA) to study and investigate data sets and describe their key characteristics, often using data visualization techniques [4]. It assists data scientists in determining how to best manipulate data sources to obtain the answers they require, making it easier for them to find patterns, test hypotheses, and verify assumptions.

5.1 Which event type dominate the task run times

To determine which event type, dominate the task run times, we need to calculate the time difference between the events. The dataset consists of 5 events for every event there is two event type start and stop. The start is when the event was started and stop is when the event was stopped. In the dataset we have don't have separate start time and stop time columns, so it will be very difficult to calculate the time taken by each event. So, for easy calculation we are separating the timestamp into two columns start time and stop time based on event type start and stop for each event. The dataset will also be reduced to half. The difference between start time and stop is calculated in seconds.

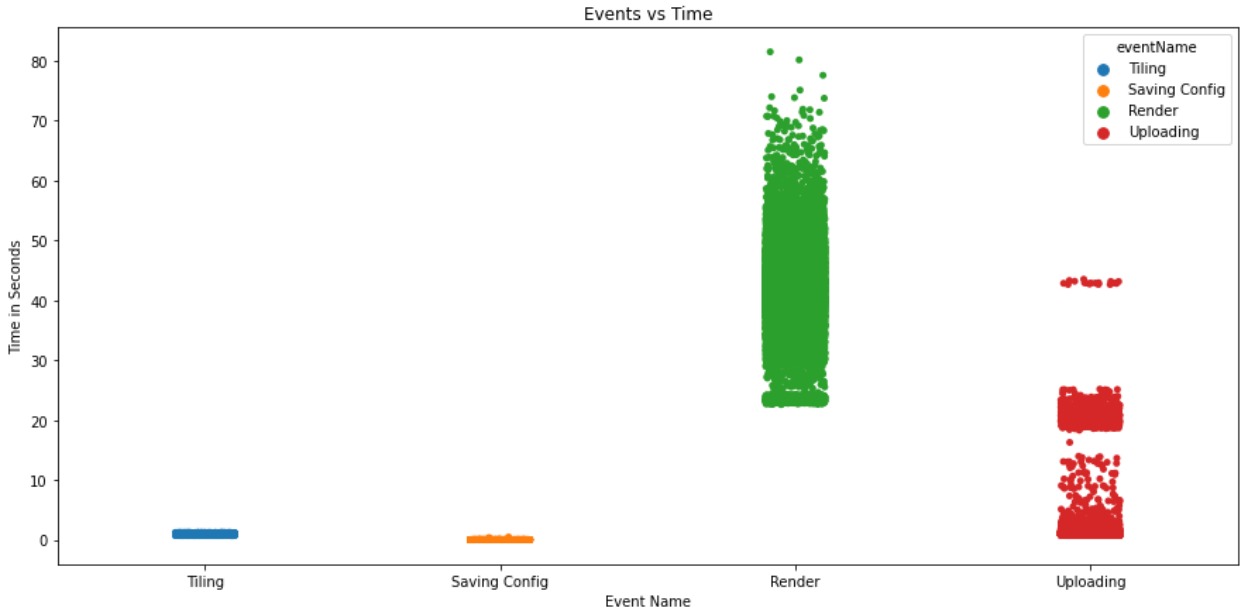


Fig 4: Events vs Time (in Seconds)

From above plot we can see that rendering the image takes more time than any other task in overall rendering process.

5.2 predicting which event type dominates the task run times for each jobId.

To predict which event type dominates the task run times in each jobId, firstly we need to know how many unique jobId (Azure jobs) are present in the dataset. But from the description of data, we already know they are three unique Azure jobs present in the application-checkpoint dataset. The three Azure jobs are:

S.No	**Azure Jobs**
1)	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705
2)	1024-lvl8-5ad819e1-fbf2-42e0-8f16-a3baca825a63
3)	1024-lvl4-90b0c947-dcfc-4eea-a1ee-efe843b698df

Table 1: Table showing Azure Job-Id's

Plotting the striplot for each Azure job seeing which event type dominates the task run times in each job.

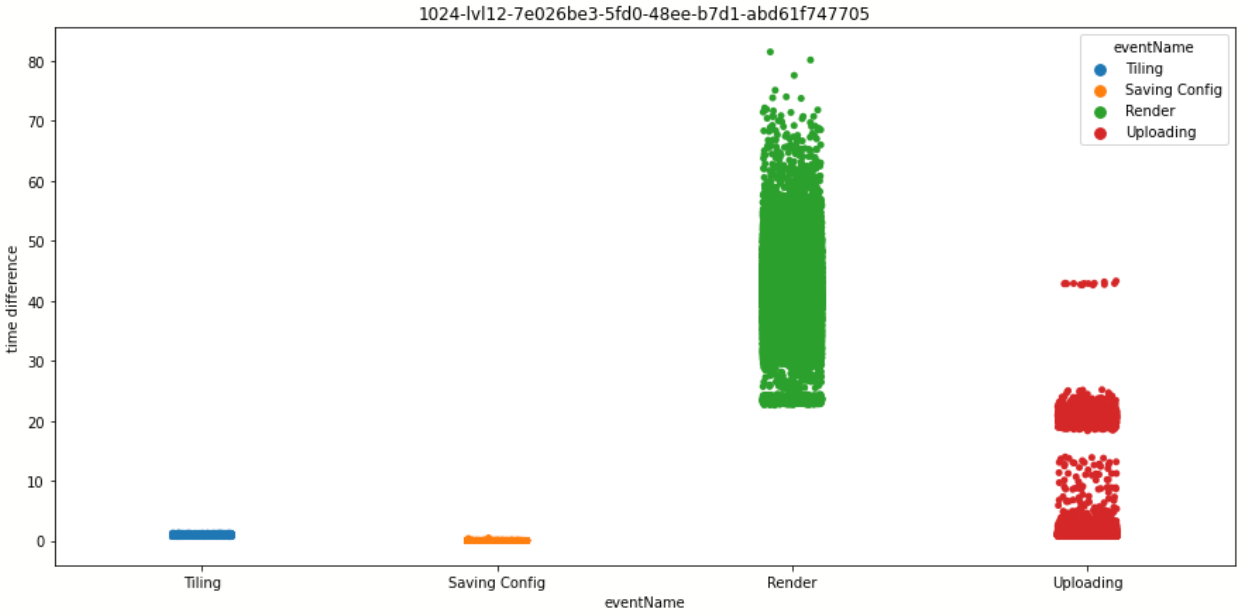


Fig 5: Events vs Time for job-id (1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705)

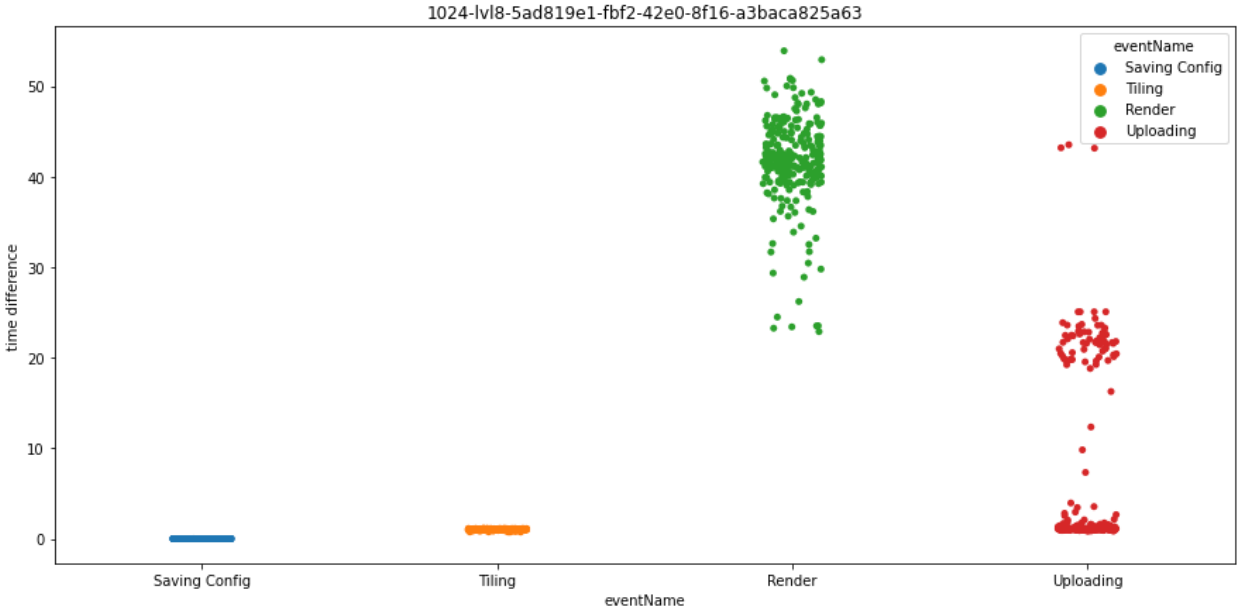


Fig 6: Events vs Time for job-id (1024-lvl8-5ad819e1-fbf2-42e0-8f16-a3baca825a63)

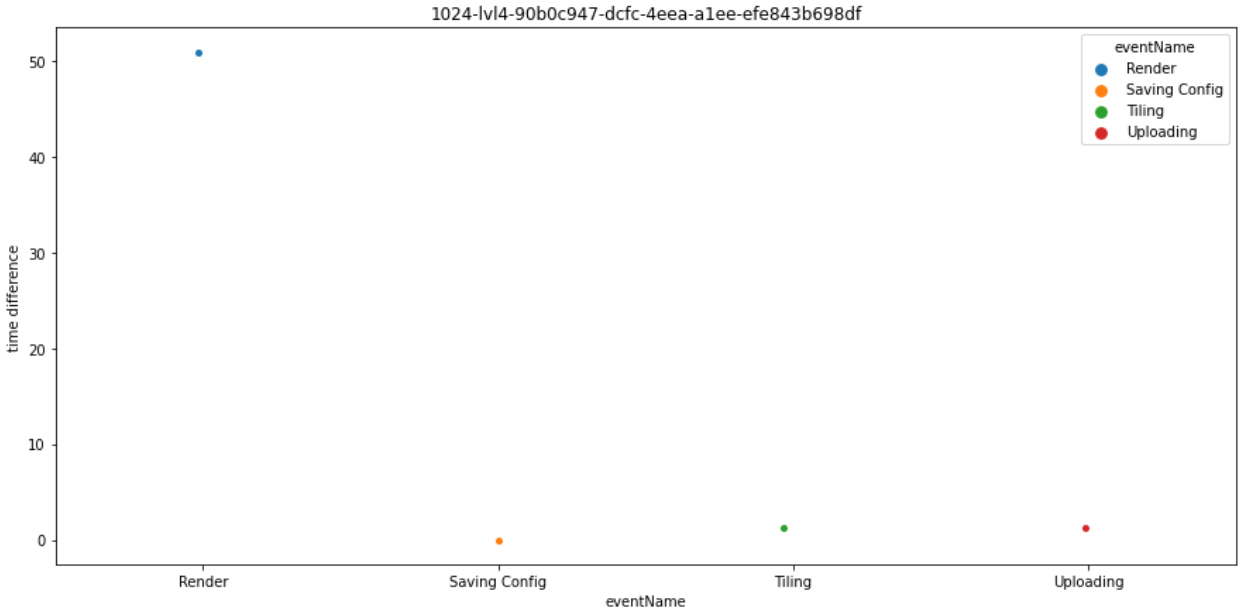


Fig 7: Events vs Time for job-id (1024-lvl4-90b0c947-dcfc-4eea-a1ee-efe843b698df)

From above plot we can again see that for each job id the render process takes more time than other events.

5.3 Which level dominates the task run time.

In this part, we perform the analysis to predict which level dominates the task runtime

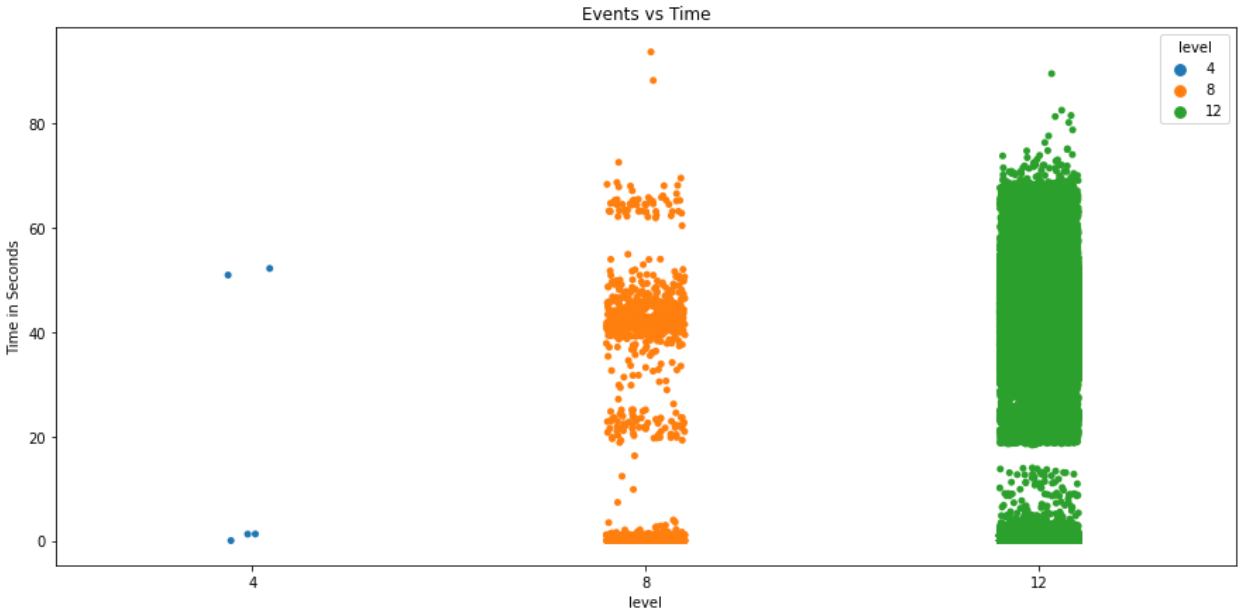


Fig 8: Level vs time

The above graph shows that level 12 dominates the task run time. Whereas level 4 and 8 has very less impact on the run time. That is, the actions performed on the image by the levels 4, 8 are very less. Most of the gpu power is utlizied in rendering level 12 that is the inner most zoomed layer of the image.

5.4 Correlation between attributes of GPU dataset

To find out the interplay (correlation between GPU temperature and performance we need to find the correlation between the GPU temperature and performance.

Correlation: It's a metric for determining how closely two variables are linked. A correlational analysis can produce three outcomes: a positive correlation, a negative correlation, or no correlation.

- A positive correlation is a two-variable relationship in which both variables move in the same direction.
- A negative correlation is a two-variable relationship in which both variables move in the opposite direction
- Zero correlation exits when there is no relationship between two variables.

	gpuSerial	powerDrawWatt	gpuTempC	gpuUtilPerc	gpuMemUtilPerc
gpuSerial	1.000000	-0.013008	-0.142388	0.001451	0.004646
powerDrawWatt	-0.013008	1.000000	0.537191	0.862673	0.870745
gpuTempC	-0.142388	0.537191	1.000000	0.505906	0.492783
gpuUtilPerc	0.001451	0.862673	0.505906	1.000000	0.966571
gpuMemUtilPerc	0.004646	0.870745	0.492783	0.966571	1.000000

Table 2: Correlation between different attributes of gpu dataset

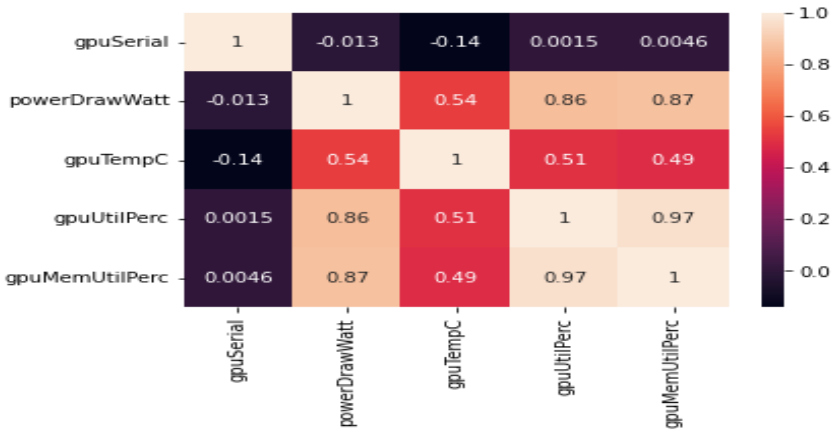


Fig 9: Heatmap representing the correlation between different attributes of gpu dataset

5.4.1 Correlation between percent utilization of the GPU (core) and percent utilization of the memory

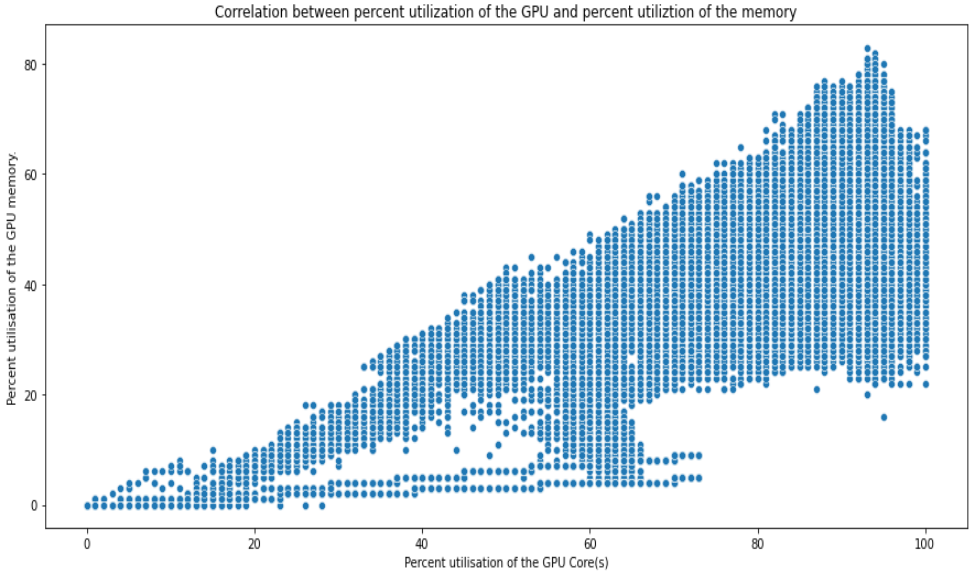


Fig 10: Correlation between Percent utilization of the GPU and percent utilization of the memory

From above we can clearly see that percent utilization of the GPU (core) and percent utilization of the memory are highly correlated with each other. As memory utilization increases percent utilization also increases.

5.4.2
Correlation between temperature and percent utilization of the GPU (core) and per cent utilization of the memory

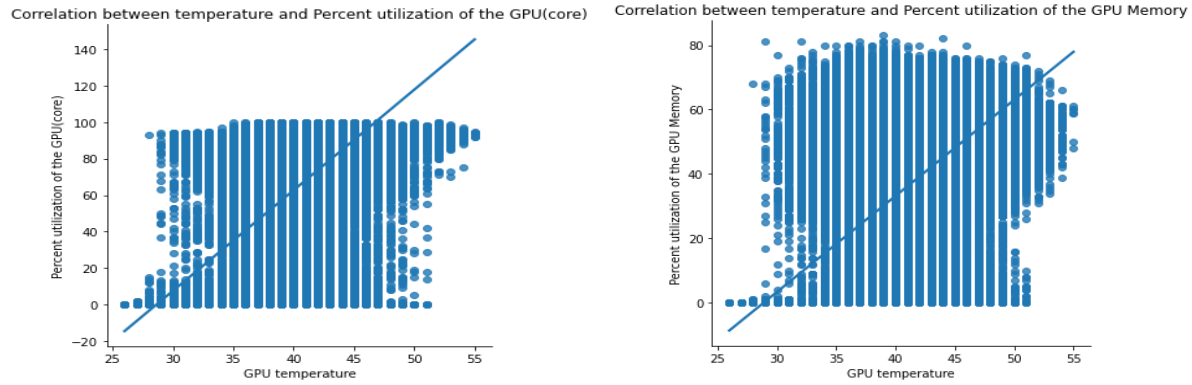


Fig 11: Correlation between temperature and percent utilization of the GPU (core) and percent utilization of the memory

GPU temperature has moderate effect on the percent of utilization of the GPU (core) and percent utilization of the memory.

5.4.3 Correlation between Power drawn and remaining attributes

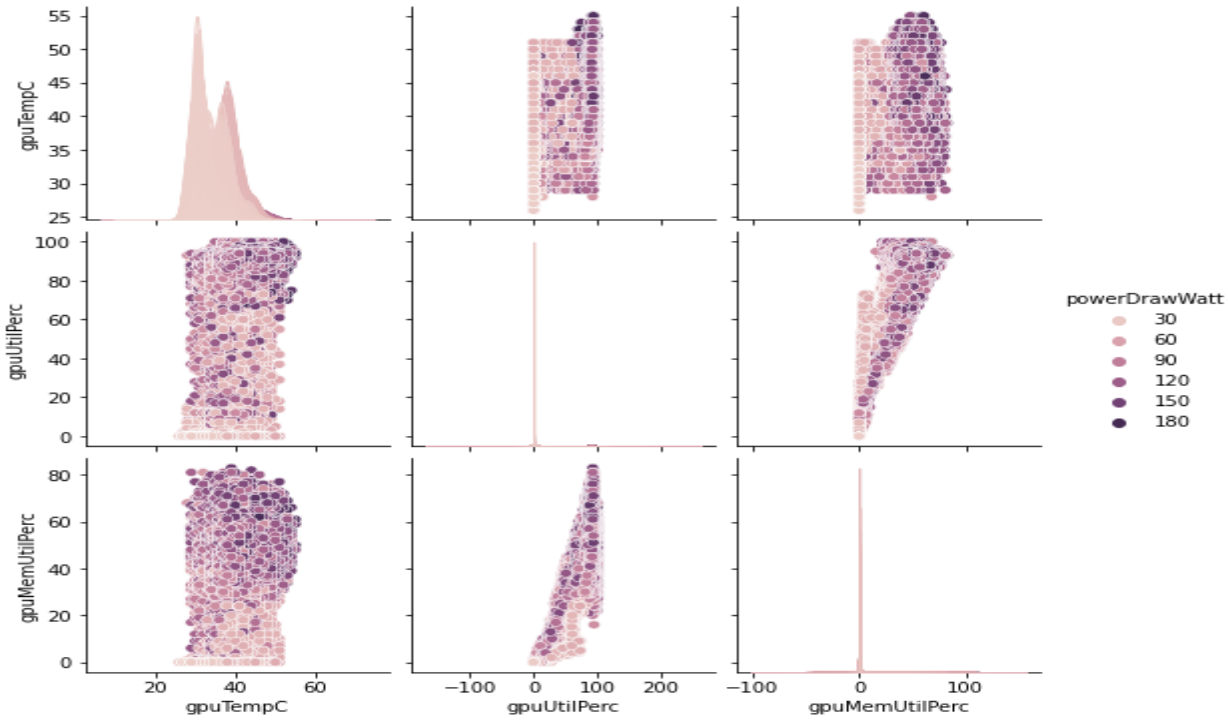


Fig 12: Pairplot to show relationship power drawn and other attributes of GPU stats

From above pairplot we can see, power drawn has a significant positive linear relationship with gpu utility percentage and gpu memory utility percentage, implying that as power drawn increases, so does gpu and memory use. It has a moderately positive relationship with gpu temperature as well.

5.5 Analyzing GPU Hardware performance by plotting box plot

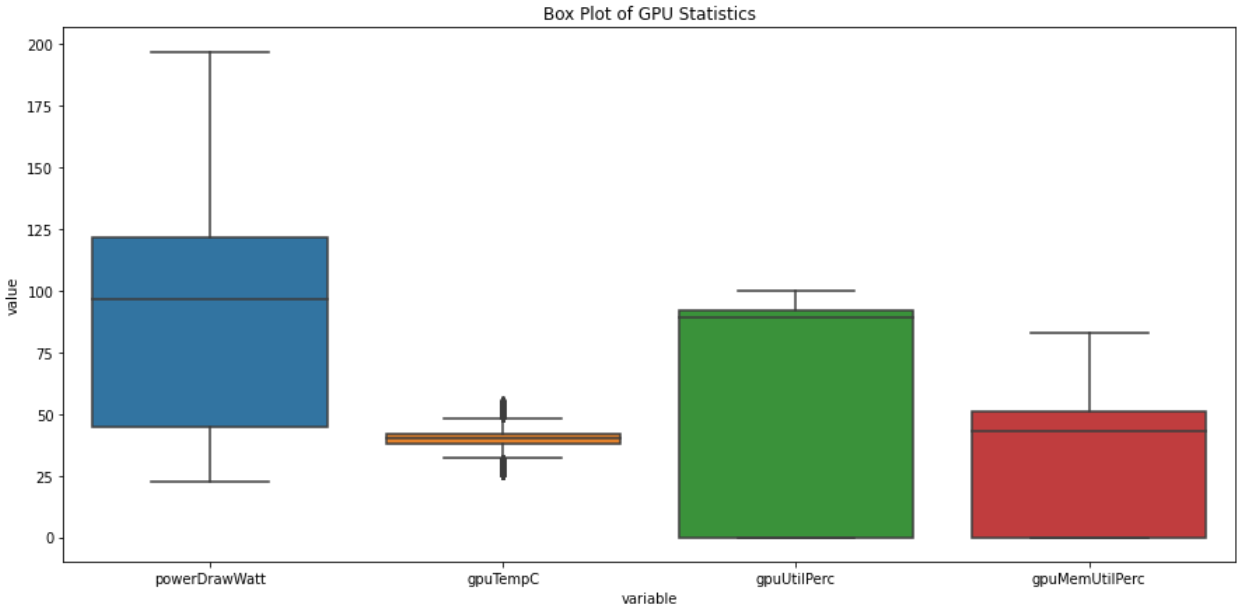


Fig 12: Box Plot of GPU Statistics

Various gpu parameters acquired during image rendering are represented in the above box plot of gpu statistics.

- **GPU Power Drawn:** Under normal conditions the power range is between 45 to 121W, with minimum 23W and maximum 200W.

- **GPU Temperature:** GPU temperate for maximum cases lies between 30-to-60-degree Celsius.

- **GPU Utility percentage:** Since the average is about 85-90 percent, we may assume that most GPUs are almost fully utilized, however, due to the wide interquartile range, certain changes are still required for more efficient handling.

- **Memory Utility percentage:** The average here is around 50%, with a smaller and narrower interquartile range, implying that either the rendering work is not memory intensive or the tasks. are not well optimised to utilise the memory.

5.5.1 Power Consumption range

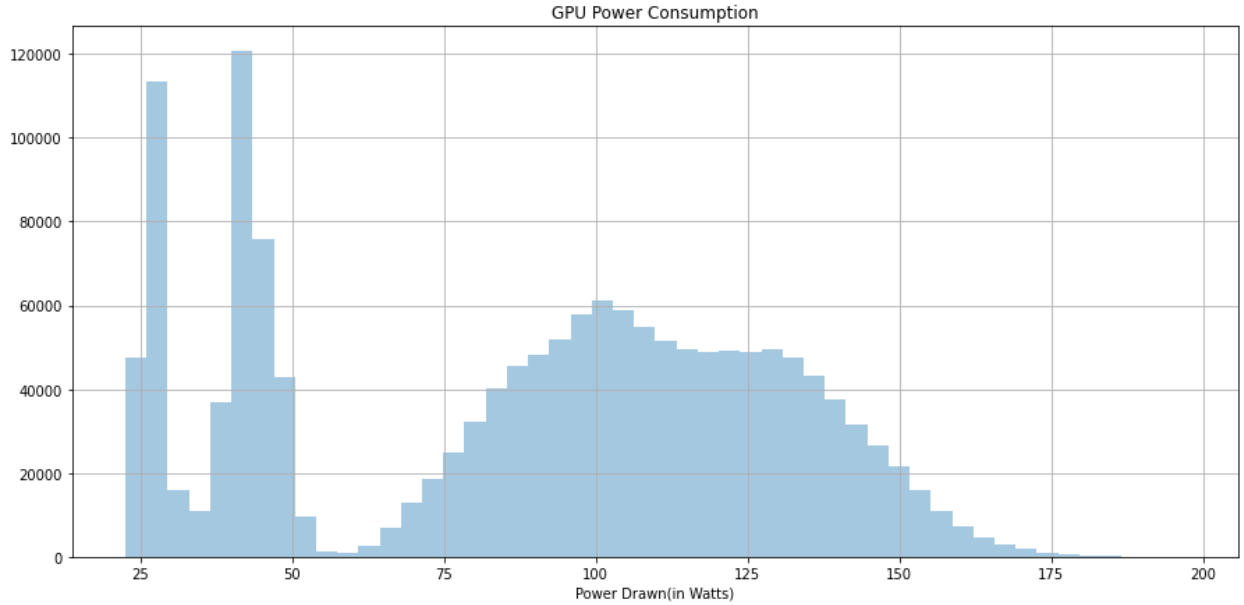


Fig 13: Power consumption Range

From above again we can see that most of the GPU consumes between 20 to 45W of power.

5.6 Correlation Between Render time and Power Drawn

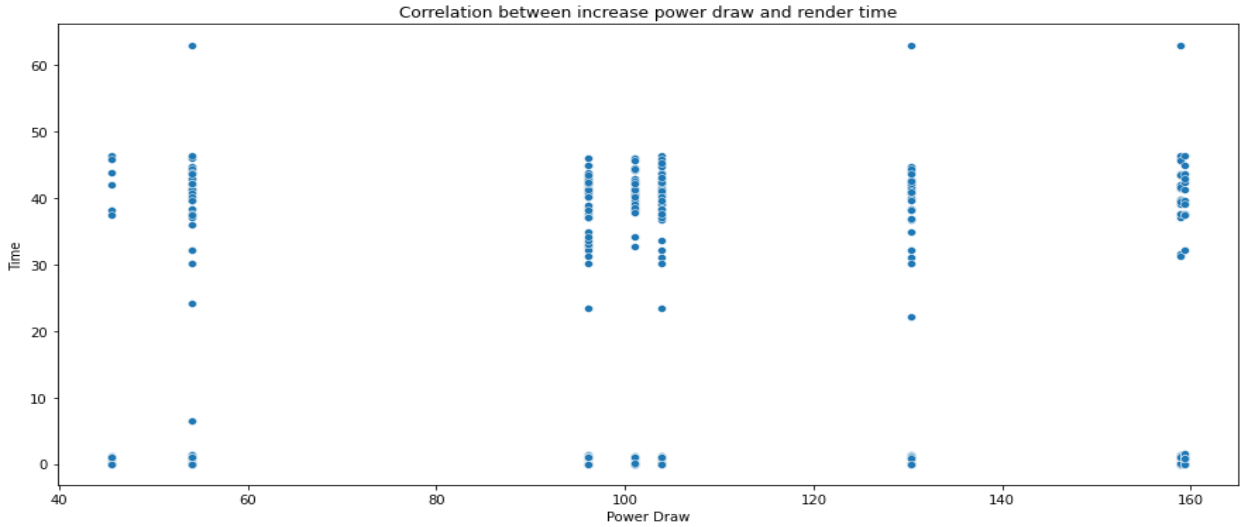


Fig 14: Correlation between increase power draw and render time

From above we can clearly see that there is no correlation between the render time and power

drawn from gpu. We can see when the power is high the rendering time is less in some cases and in some cases it's opposite.

6 Conclusion, Results and Future Work

Summarization of above Exploratory Data Analysis on TersoScope dataset:

The GPU spends the majority of its time rendering when compared to other events (tiling, uploading, saving configuration). It is the GPU's most critical task. GPU power consumption and temperature readings were good. While plotting the correlation between different GPU stats we find out that GPU temperature has moderate effect on the percent of utilization of the GPU (core) and percent utilization of the memory, percent utilization of the GPU (core) and percent utilization of the memory are highly correlated with each other. There was no proper correlation between the render time and power drawn by GPU.

6.1 Future Work

We can use predictive technique to analyze the GPU performance and reliability.

7 Learning

The project taught me how to use a number of tools and improved my understanding of how to evaluate the performance of a system. It has helped me better understand how to use CRISP-DM. Using the Python language and a literate programming framework, I've acquired a variety of analytical strategies for performing data analysis and obtaining relevant information. In the project, I used Jupyter Notebooks and popular libraries like Pandas (for data processing, data cleansing), matplotlib, and seaborn (for data visualization), as well as GitHub for version control, all of which helped me refine my skills in using these tools. I was also able to improve my report writing and data science project management skills as a result of the assignment.

8 Reference

[1] https://www.microsoft.com/en-us/research/publication/data-intensive-science-terapixel-modisazure-projects/

[2] Holliman, N.S., Antony, M., Charlton, J., Dowsland, S., James, P. and Turner, M., 2019. Petascale cloud supercomputing for terapixel visualization of a digital twin. *IEEE Transactions on Cloud Computing*.

[3] https://github.com/NewcastleDataScience/StudentProjects202122/blob/master/TeraScope/Summary.md

[4] https://www.ibm.com/uk-en/cloud/learn/exploratory-data-analysis