Journal of Science and Technology
ISSN: 2456-5660 Volume 5, Issue 4, July-August 2020

www.jst.org.in *DOI: https://doi.org/10.46243/jst.2020.v5.i4.pp230-237*

Modular Microservice based GPU Utilization Manager with Gunicorn

Remya A. R[1]., Arun A. Balakrishnan[2], Suraj Kamal[3], Satheesh Chandran C[4]., Supriya M. H.[5]
[1,2,3,4,5]*(Department of Electronics, Cochin University of Science and Technology, Kochi, India – 682022)*
[1]*Corresponding Author: remyacusat@gmail.com*

To Cite this Article
Remya A. R., Arun A. Balakrishnan, Suraj Kamal, Satheesh Chandran C., and Supriya M. H., "Modular Microservice based GPU Utilization Manager with Gunicorn", Journal of Science and Technology, Vol. 5, Issue 4, July-August 2020, pp230-237

Article Info
Received: 07-04-2020 Revised: 01-07-2020 Accepted: 10-07-2020 Published: 13-07-2020

Abstract:*Graphics processing unit (GPU) is a computer programmable chip that could perform rapid mathematical operations that can be accelerated with massive parallelism. In the early days, central processing unit (CPU) was responsible for all computations irrespective of whether it is feasible for parallel computation. However, in recent years GPUs are increasingly used for massively parallel computing applications, such as training Deep Neural Networks. GPU's performance monitoring plays a key role in this new era since GPUs serve an inevitable role in increasing the speed of analysis of the developed system. GPU administration comes in picture to efficiently utilize the GPU when we deal with multiple workloads to run on the same hardware. In this study, various GPU-parameters are monitored and help to keep them in safe levels and also to keep the improved performance of the system. This study, also delivers the GPU monitoring protocol as a microservice that is deployed to Gunicorn production server.*

Keywords*:Graphics processing unit (GPU), central processing unit (CPU), NVIDIA, docker, microservices, orchestration, high-performance computing (HPC), inter process communication (IPC)*

I. Introduction

GPUs gave way to a boom in the AI industry by providing parallel processing architecture. Nowadays, GPUs are able to share the workload of CPUs and train deep neural networks for AI applications and thus become the key to AI and supercomputing applications. GPU[1], the specialized electronic circuit often present on a video card or embedded on the motherboard or CPU is designed to fast deploy and accelerates the applications running on its CPU. Importance of GPUs arise with the enhancement of more advanced systems that deal with AutoCAD and other high-performance supercomputing technologies such as rendering realistic 3D computer graphics. [2],[3]General-purpose CPUs were in place of dealing the mathematical calculations till date were overwritten by the GPUs which can handle parallel-computing architecture by subdividing the task into smaller processing units that run simultaneously in the machine.

Nowadays, the use of GPUs can be seen in embedded systems, game consoles, personal computers, mobile phones and other mobile devices, data centers, workstations.

NVIDIA and AMD are the two main holders in the graphics card field of desktop and laptop graphics applications. Others include Intel's own Iris Plus, UHD integrated GPUs to work without dedicated graphics whereas Qualcomm and MediaTek provide GPUs for mobile devices with an embedded GPU on the same CPU chip.

Assessment of GPU utilization is of great importance to deal with better performance system characteristics. GPU monitoring often associated to three main dimensions:

- GPU-dimension metrics
- Instance-dimension metrics
- Group-dimension metrics

Developing GPU monitoring and managing as a microservice enables us to deploy the application in any platform to continuously monitor the GPU hardware connected to it. In this study NVIDIA GPU is considered and developed an application that fetches GPU metrics such as temperature, fan speed, clock, memory usage etc. and displays the measures for the users. Also, it is developed in such a way that, if a particular GPU is exhausted, for example its temperature goes beyond a specific limit, the clock speed or the effective power utilization will be throttled to bring the temperature back to the safe limits although there are hardwired measures to safeguard GPUs against thermal damage. Thereby helping the entire system from damages associated to overloading or overheating and also helps to identify any possibilities of defect or break down of the fan, memory usage etc.

Aim of this study is to address the GPU utilization monitoring to guarantee the improved system performance. Running high-performance applications increase the usage of GPU and as a result GPU-dimension metrics such as temperature, memory usage, clock cycles also increase impacting performance measures. This leads to the importance of developing a system that can monitor GPU utilization and control the parameters if it goes beyond a specific limit. The field of HPC currently lacks an interactive microservice that can address monitoring of GPU performance also has been considered as a motivation for the development and deployment of this containerized GPU management application.

Proposed scheme is a novel study in the field of GPU monitoring and controlling. For processes that runs for hours, GPU won't delay due to over use as this microservice supports monitoring and controlling the GPUs in background.

II. Existing GPU Monitoring Tools

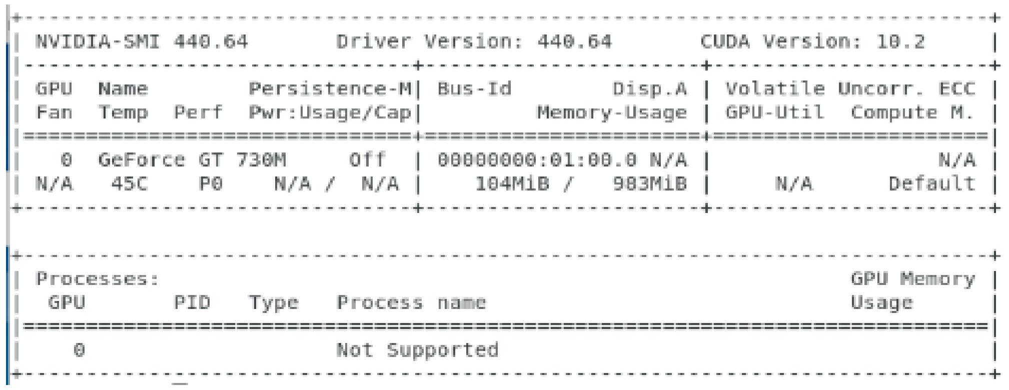
Looking at the literature, it is seen that there are very few studies associated with the development of GPU monitoring software. [4]–[6]NVIDIA, the most commonly used GPU platform itself offers some APIs for monitoring its performance. GPU Shark, is another open-source lightweight GPU monitoring tool, based on ZoomGPU, for NVIDIA GeForce and AMD/ATI Radeon graphics cards. Following sections discuss the available works in this area, an effort is given to cover most of the available studies but may not include all of them.

A. NVIDIA GPU Monitoring Tools

a) **Monitoring NVIDIA GPUs using API** - a C-based API for monitoring and managing NVIDIA GPU devices

NVIDIA official APIs[5] are NVIDIA Management Library (NVML[7]) and NVIDIA's core software development kit (NVAPI).

NVML – This API provides a reference metrics by using nvidia-smi command-line utility and it is used when validation of API usage precision is desirable. When performing this command, the resulting metrics contains envisioned fan speed for all fans present on the device (in % of max RPM); real chip temperature (in oC); real power usage and power cap (in W) for GPU, memory and circuitry; real memory utilization (in %); and real GPU utilization (in %). Figure 1 demonstrate this result more clearly. This API is available in both Windows and Linux-based systems. Other advantages of NVML are its usage of the most recent drivers and shared library. Whereas the absence of header files with data types as well as function definitions and lacking manual fan control permissions are its disadvantages.



```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.64       Driver Version: 440.64       CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GT 730M     Off  | 00000000:01:00.0 N/A |                  N/A |
| N/A   45C    P0    N/A /  N/A |    104MiB /   983MiB |     N/A      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0                    Not Supported                                       |
+-----------------------------------------------------------------------------+
```

Figure 1: nvidia-smi

NVAPI – This API is solely for Windows 32- and 64-bit architectures but both static and dynamic libraries are available. It measures absolute fan speed in RPM and the temperature for each sensor, and also the functions are loaded using memory offset.

Thus, GPU monitoring API collects core metrics such as fan speed, temperature, device power consumption, GPU utilization, and GPUs memory utilization.

b) Monitoring NVIDIA GPUs using DCGM - a set of tools for monitoring and managing NVIDIA GPUs in cluster environments. DCGM performs active health monitoring, diagnostics, system validation, policies, power and clock management, group configuration and accounting on each host system.

DCGM exporter[8] is a container to help gather GPU metrics which can be pulled from the Docker Hub using the command ‘docker run -d --gpus all --rm -p 9400:9400 nvidia/dcgm-exporter:latest’, after that the URL ‘localhost:9400/metrics’ can be checked to see the metrics values. Grafana Dashboard is a dashboard graphical display of GPU metrics read by the dcgm-exporter container.

B. GPU Shark

GPU Shark[9], [10] a free GPU monitoring tool that can display the clock speeds, fill rates, GPU fan speed, memory usage, power consumption, performance states (PStates), etc. for every GPU but this application works only for Microsoft Windows. It works for NVIDIA GeForce and AMD/ATI Radeon graphics cards and the information display comes in two modes such as simplified or default mode and detailed mode. In simplified mode, only the significant hardware information such as graphics card name, GPU and PCB temperatures, and GPU, memory, and shader clocks are shown but in the detailed mode, all data is available such that GPU codename, driver and bios version, device ID, performance states, temperatures etc.

C. GPU Caps

GPU Caps Viewer[11] is an OpenGL-OpenCL graphics card utility for Windows XP and Vista platforms (32- and 64- bits). GPU Caps also provide information on graphics configuration, NVIDIA CUDA level support, GPU core temperature, CPU information, and also graphics card validation but apart from GPU Shark, this tool mainly focused on the main graphics card that runs on OpenGL and Direct3D apps. This software is not simple as GPU Shark since it comes with several DLL and demos data files.

D. GPU-Z

GPU-Z[12], [13] is also an open-source lightweight GPU monitoring utility from TechPowerUp. This software also supports NVIDIA, AMD, ATI and Intel graphics devices by displaying information on adapter, GPU, and display information. This tool also works only with Windows versions such as XP/Vista, Windows 7, Windows 8, Windows 10 (32 and 64- bits) even though it has multi-GPU support.

E. Other Studies

GPU temperature monitoring is most important when critical operations are in place since it can prevent failure and identify associated problems. [14]MSI's Afterburner tool helps to monitor GPU temperature, core clock and fan control and allows it to monitor even during the game period. But its interface seems to be quite confusing when compared to other packages. EVGA Precision X1 is another application that works with GeForce RTX 20-series GPUs that has a simple interface compared to the Afterburner and provides easy access and accurate readings to track GPU temperature. OpenHardware Monitor is another option to check for the hardware’s including GPUs temperature, fan speed, frequency, and voltage values but with compromise on its estimation accuracy. SpeedFan, also offers a simple interface to monitor the hardware information and with some controller on fan speed control. HWInfo[15] is a PC monitoring program and its Sensors icon displays information on GPU such as GPU version, temperature, core voltage, fan, power, clock, memory clock, video clock, core load etc. Proactive monitoring and performance troubleshooting are made possible with Goliath Performance Monitor[4], [16] for NVIDIA GPU by identifying and preventing overloaded GPUs and performing proper allocation strategies.

Windows 10, now has a new feature to monitor its GPU performance[17]–[19]. In order to achieve this, activate the ‘GPU’ and ‘GPU Engine’ options in the process tab of task manager. Percentage of GPU performance as well as overall GPU resource usage can be verified though this interface.

Better utilization of resources can be guaranteed by keeping track of GPU usage statistics[20]. On VM instances, GPU metrics can be monitored using Cloud Monitoring by downloading the GPU metrics reporting scripts and enabling the GPU monitoring scripts. Then the Google Cloud Console□Monitoring or google Cloud Console□Metrics Explorer□gpu_utilization, gives information on the GPU statistics. Improving GPU usage for training deep learning models by monitoring and measuring its metrics is described in [21]. wandb python package is suggested in this work to track CPU, GPU, and memory usage and other system metrics. Glances[22] is one of the cross-platform real-time system monitoring tool that gathers information such as temperature, memory, load etc.

especially for CPU. Psensor[23] is another temperature monitoring tool for graphical hardware and is GTK+ based software.

A case study on GPU orchestration using docker is presented in [24] that talks on the importance of GPU orchestration for supporting the optimal GPU utilization. docker-nvidia is the runtime used in this study to mount the underlying docker to the container. Consuming GPUs across different Kubernetes versions are presented in the documentation[25] that contains experimental support for managing GPUs such as AMD and NVIDIA across several nodes. This study covers how efficient GPUs can be scheduled to work with workloads but with some limitations such as specify GPUs in the limits section, no GPU sharing among Containers and Pods, and Containers request one or more GPUs but not a fraction of it. This article [26] discusses, orchestrating GPU-Accelerated workloads on Amazon ECS. In this scheme, it is said that in order to develop GPU usage to train neural networks, a Docker image configured with NVIDIA CUDA drivers must be set in Amazon ECR to allow the container to communicate with the GPU hardware.

III. Proposed GPU Monitoring System

GPU performance monitoring and management is an inevitable part when working with HPC applications to keep the applications running without any performance degradations or failure. Proposed scheme envisions monitoring and controlling the GPU in the background since it is deployed as microservice.

From the architectural diagram[27], [28] (Figure 2), it is understood that the proposed GPU monitoring[29], [30] application is built on top of the nvidia: cuda base image on a Linux machine which has NVIDIA drivers installed on it.

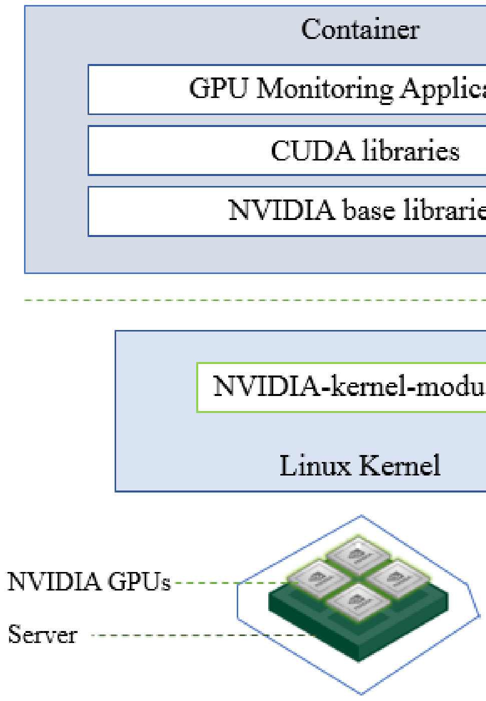


Figure 2: Basic Architecture

This containerized GPU application first checks if the system supports control clock or power management, that decides the workflow. Once this application is invoked in the system, initial process monitors the available GPUs, fetches its performance measures, put them in a FIFO buffer; second process listens the FIFO queue and send these GPU measures into the dashboard display. It is implemented in such a way that; the entire process communicates through inter process communication and is independent of any processes that make use of the available GPUs. A callback mechanism is also incorporated so as to monitor the GPUs in every second which can be modified in the program. Once this service is started, it will automatically run in the background and controls the GPU clock or power, even if the user doesn't watch it.

A. Inter process Communication

GPU monitoring system comprises the design and development of two processes that communicate with each other through inter process communication (IPC). NVML API module is designed to monitor and control and the

Dashboard module to fetch buffer/queue and display in the dashboard. First step is to identify the number of available devices and GPUs associated with it.

a) NVML API module

This module is included to interact with the NVIDIA GPU devices for monitoring and management of its various states. It also provides a direct access to the queries and commands visible through nvidia-smi. [31]–[34]Multiprocessing Queue object is employed in this module which is a FIFO data structure used to pass data between different process without any potential corruptions on it. [35]–[37]Doubly Ended Queue (deque) is implemented from the collections module to handle the quick data processing activities such as quick append and pop operations on both ends of the container to support the parallel processing.

Based on the available gpus index, the process acquires the handle for that particular device so as to query its Query-able GPU statistics[38] such as power usage nvmlDeviceGetPowerUsage (nvmlDevice_t device, unsigned int* power), fan speed (nvmlDeviceGetFanSpeed (nvmlDevice_t device, unsigned int* speed)), temperature (nvmlDeviceGetTemperature (nvmlDevice_t device, nvmlTemperatureSensors_t sensorType, unsigned int* temp)), clock info (nvmlDeviceGetClockInfo (nvmlDevice_t device, nvmlClockType_t type, unsigned int* clock)), supported memory clocks (nvmlDeviceGetSupportedMemoryClocks (nvmlDevice_t device, unsigned int* count, unsigned int* clocksMHz)), supported graphics clocks (nvmlDeviceGetSupportedGraphicsClocks (nvmlDevice_t device, unsigned int memoryClockMHz, unsigned int* count, unsigned int* clocksMHz)), are the GPUs states retrieved here and appended to the deque. After the first cycle, applications clock (nvmlDeviceResetApplicationsClocks (nvmlDevice_t device)) and the power management limits (nvmlDeviceSetPowerManagementLimit (nvmlDevice_t device, unsigned int limit) are set back to the default value and thus helps to clear any stale throttle to put them ready in the queue.

b) Dashboard module

Dashboard[39] is the graphical user interface used in this Dash application. In this python module, Dash is imported and activated as the first step, data preparation is given to the NVML API handler, and another process is defined to fetch the GPU measurements and then the visualizing charts defined in the application layout displays the performance measurements. This module listens to the FIFO buffer and send these GPU performance measures to the dashboard display. It acquires basic information such as gpus, threshold temperature, wait duration etc. from the JSON file then the NVML API comes into action which is described in the following section.

GPU Watcher dashboard plots some of the fetched GPU data such as GPU Temperature as Scatter Graph and histogram, GPU Control Settings that is its Temperature Limit and Power Usage in another scatter plot and bar chart. Figure 3 illustrates the basic flow of the entire GPU watcher application in which the gpu_manager, the main program, calls the actual dash application and invokes the Flask server.

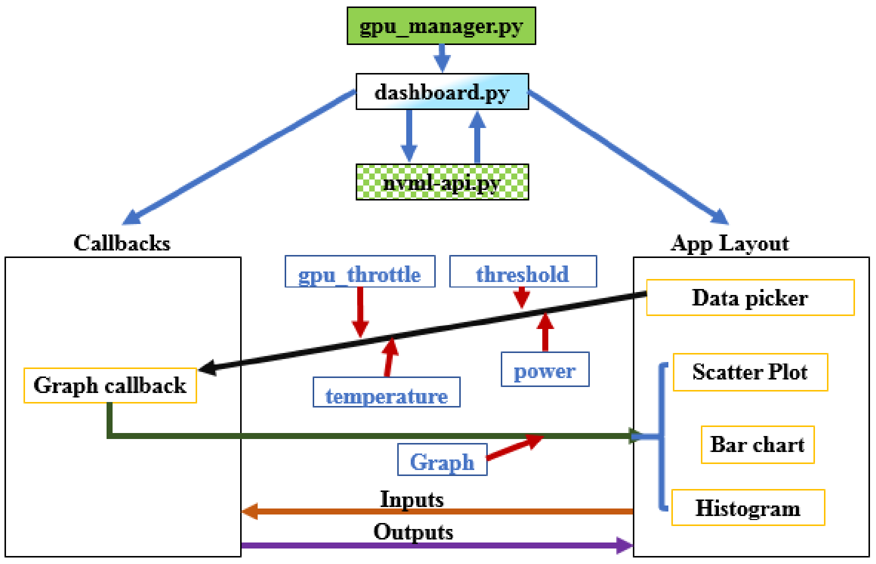


Figure 3: Data flow diagram of GPU Watcher

B. Dockerizing the GPU Watcher Application

Now that GPU Watcher application is up and running. Next step is to containerize the application in docker. For that create the requirements text file that contains the required dependencies and Dockerfile which is responsible to setup the application with nvidia/cuda:10.0-base-ubuntu18.04 as the base image. Then build the docker image from the program directory using the following command in the terminal:

```
docker build --tag=gpumgr •
```

Upon execution of this command, a docker image named gpumgr is created as a layered structure following the Dockerfile commands. Then run the application using the following command:

```
docker run --runtime=nvidia --privileged=true --rm -it -p 7000:7000 gpumgr
```

Now the application runs but with a warning on development server and can be accessed at http://localhost:7000or http://0.0.0.0:7000/

C. Deploying the GPU Watcher application with Gunicorn

Gunicorn[40]–[43] is a commonly used WSGI server for deploying python applications in production. Gunicorn's main process or master process starts one or more worker processes. It has heartbeat system which ensures the workers are alive and restart them if the workers die. Same project layout can be used but with some updates on gunicorn dependency in the requirements file and Dockerfile as well as explicitly mention the Flask app server.

- Once these changes are completed, docker image can be build using the same command:

```
docker build --tag=gpumgrprod •
```

- Docker image created with name gpumgrprod, can now be run in the Gunicorn server using the following commands:

```
docker run --runtime=nvidia --privileged=true --rm -it -p 7000:7000 gpumgrprod
```

Now the application runs without any warnings in the terminal. Terminal shows Starting gunicorn 20.0.4 and the application can be accessed at http://0.0.0.0:7000/

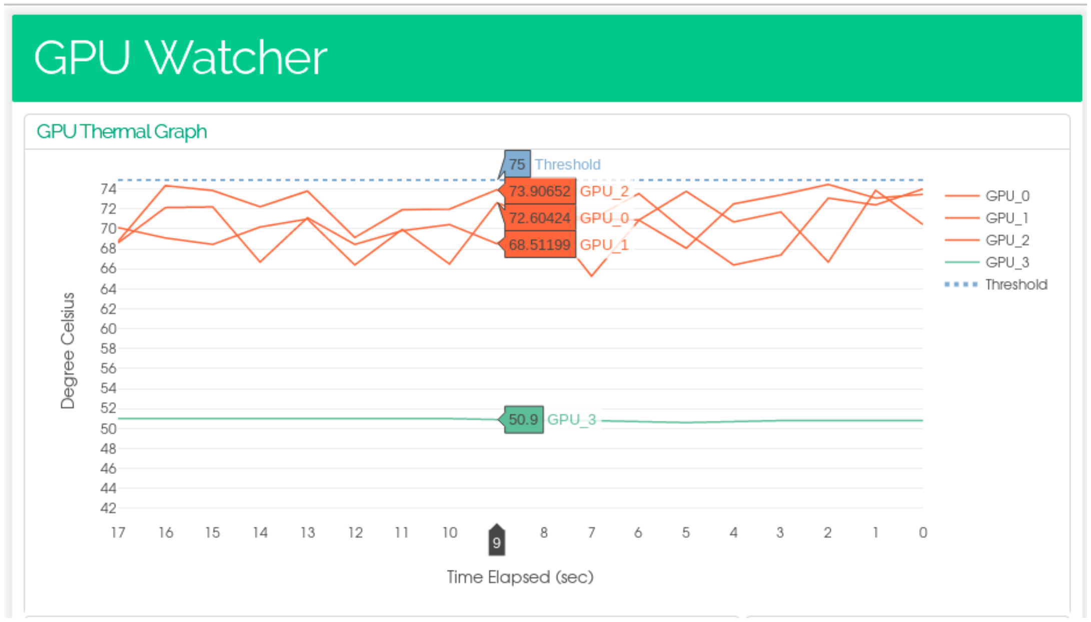


Figure 4: GPU Watcher – Thermal Graph

From the Dashboard display (Figure 4 and Figure 5), it is easier to verify GPU performance measures such as GPU Temperature, GPU control settings - Threshold, GPU power usage as a scatter graph and bar charts. With this information it is easier to identify the current statistics of GPU, so that performance characteristics of the entire system can be guaranteed to avoid system failure, performance degradation, entire system crash etc.

To discuss, deploying this GPU monitor application in production environment, starts with Gunicorn's main process which in turn starts worker processes and assigns loads to them. The program checks for the available number of GPUs and its monitoring tasks are divided among the workers according to the number of GPUs. Each active worker process is efficiently utilized by engaging their query-response waiting time for the querying of other

GPUs or statistics remaining not attended. Thus, efficiency can be guaranteed with the production deployment providing competent GPU performance monitoring and management.

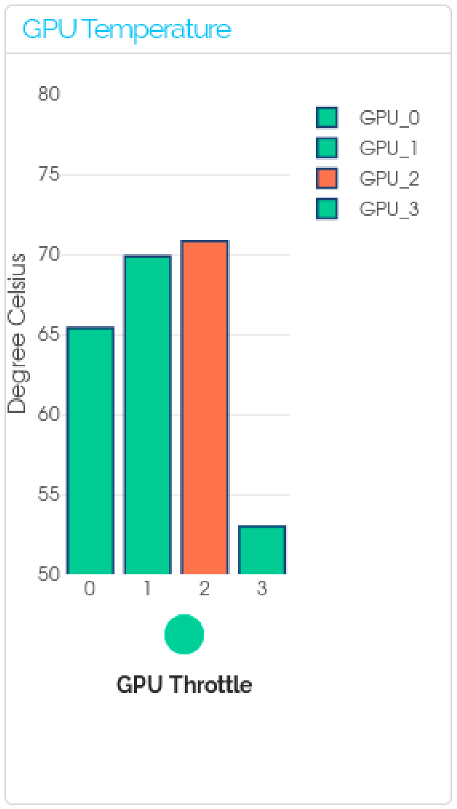


Figure 5: GPU Watcher – Temperature

IV. Conclusions and Future Scope

This work brings a GPU performance monitoring tool that is integrated as a docker container deployed in production. Since it is developed as a microservice, it will be easier for any user to pull this image from the docker hub, (once it is pushed) and run it on their machine to keep track of the GPU performance. By doing this, the user has a hold on the GPU performance characteristics so as to identify its temperature and power usage. Having knowledge on these parameters helps to identify issues such as fan failure, or failure of any components associated with it and over use of any specific components since all these may result in a substantial hike in the temperature.

This method can be extended to Kubernetes, one of the container orchestration tools employed in deploying and managing workloads related to the hyperparameter optimization search system for deep neural networks[21], [29]. That is the system can be extended to handle the parallel processing by avoiding deadlock situations in sharing GPU resources in such a way that, Master node assigns loads to the load-runners with the requested number of GPUs and schedules the jobs automatically depending on the availability of GPUs. Thus, deploying the GPU monitoring scheme has a vast impact on the HPC studies and is a trending area since it has more impact on the performance characteristics of the entire system.

References

[1]. A. Shepherd, "What is a GPU?," What is a GPU? | IT PRO. https://www.itpro.co.uk/hardware/30399/what-is-a-gpu.
[2]. B. Caulfield, "What's the Difference Between a CPU and a GPU?" https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/.
[3]. "WHAT IS GPU COMPUTING?" https://www.boston.co.uk/info/nvidia-kepler/what-is-gpu-computing.aspx.
[4]. "NVIDIA VIRTUAL GPU MANAGEMENT & MONITORING." NVIDIA - Whitepaper.
[5]. O.Oblovatnyi, "Monitoring Nvidia GPUs using API." https://medium.com/devoops-and-universe/monitoring-nvidia-gpus-cd174bf89311.
[6]. "NVIDIA System Monitor." https://www.nvidia.com/en-us/drivers/system-monitor/.

[7]. "NVIDIA Management Library (NVML)." https://developer.nvidia.com/nvidia-management-library-nvml.
[8]. "Tools for monitoring NVIDIA GPUs on Linux." https://github.com/NVIDIA/gpu-monitoring-tools.
[9]. "GPU Shark." https://www.ozone3d.net/gpushark/.
[10]. "GPU-Shark 0.1.0: Simple GPU Monitoring Utility." https://www.geeks3d.com/20100517/gpu-tool-gpu-shark-0-1-0-simple-gpu-monitoring-utility/.
[11]. "GPU Caps Viewer." https://www.ozone3d.net/gpu_caps_viewer/.
[12]. "TechPowerUp GPU-Z." https://www.techpowerup.com/gpuz/.
[13]. G. Gear, "Monitor Your GPU on Windows with GPU-Z by TechPowerUp." https://blogs.windows.com/windowsexperience/2013/08/01/monitor-your-gpu-on-windows-with-gpu-z-by-techpowerup/.
[14]. C. Warmenhoven, "Monitoring Your GPUs Temperatures at All Times." https://www.pugetsystems.com/labs/support-software/Monitoring-Your-GPUs-Temperatures-at-All-Times-1399/.
[15]. B. Chacos, "How to check your graphics card's GPU temperature." https://www.pcworld.com/article/3396643/how-to-check-your-graphics-card-gpu-temperature.html.
[16]. "NVIDIA GPU Monitoring & Troubleshooting." https://goliathtechnologies.com/software/goliath-performance-monitor/infrastructure/nvidia/.
[17]. B. Burgess, "How to Monitor GPU Performance on Windows 10." https://www.groovypost.com/howto/monitor-gpu-performance-on-windows-10/#:~:text=To%20monitor%20the%20overall%20GPU,like%20encoding%20videos%20or%20gameplay.
[18]. M. HUCULAK, "How to track GPU performance data on Windows 10." https://www.windowscentral.com/how-track-gpu-performance-data-windows-10.
[19]. C. Hoffman, "How to Monitor GPU Usage in the Windows Task Manager." https://www.howtogeek.com/351073/how-to-monitor-gpu-usage-in-the-windows-task-manager/.
[20]. "Monitoring GPU performance." https://cloud.google.com/compute/docs/gpus/monitor-gpus.
[21]. L. Biewald, "Monitor and Improve GPU Usage for Training Deep Learning Models." https://towardsdatascience.com/measuring-actual-gpu-usage-for-deep-learning-training-e2bf3654bcfd.
[22]. A. Kili, "4 Useful Tools to Monitor CPU and GPU Temperature in Ubuntu." https://www.tecmint.com/monitor-cpu-and-gpu-temperature-in-ubuntu/.
[23]. "Psensor – A Graphical Hardware Temperature Monitoring Tool for Linux." https://www.tecmint.com/psensor-monitors-hardware-temperature-in-linux/.
[24]. P. Esztári, "CASE STUDY: GPU ORCHESTRATION USING DOCKER." https://codingsans.com/blog/gpu-orchestration-using-docker.
[25]. "Schedule GPUs." https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/.
[26]. C. Barclay, "Orchestrating GPU-Accelerated Workloads on Amazon ECS." https://aws.amazon.com/blogs/compute/orchestrating-gpu-accelerated-workloads-on-amazon-ecs/.
[27]. A. Gray, "GPU Architecture." .
[28]. K. Klues and Y. Li, "Supporting GPUs in Docker Containers on Apache Mesos," [Online]. Available: https://docs.google.com/presentation/d/1FnuEW2ic5d-cpSyVOUMfUSM7WxJlZtTAAWt2dZXJ52A/edit#slide=id.g16e6bdb9dc_1_0.
[29]. "Optimize Analysis Performance with a GPU." https://www.microfocus.com/documentation/idol/IDOL_12_0/MediaServer/Guides/html/English/Content/Getting_Started/ParallelProcessingAnalysisGPU.htm.
[30]. A. B. Sharma, "Parameters for Evaluating Deep Learning Program's GPU Performance." https://mc.ai/parameters-for-evaluating-deep-learning-programs-gpu-performance/.
[31]. E. Vaati, "Introduction to Multiprocessing in Python." https://code.tutsplus.com/tutorials/introduction-to-multiprocessing-in-python--cms-30281.
[32]. "multiprocessing Basics." https://pymotw.com/2/multiprocessing/basics.html.
[33]. E. Forbes, "Python Multiprocessing Tutorial." https://tutorialedge.net/python/python-multiprocessing-tutorial/.
[34]. "multiprocessing - Process-based 'threading' interface." https://docs.python.org/2/library/multiprocessing.html.
[35]. "Deque." https://pymotw.com/2/collections/deque.html.
[36]. "Deque in Python." https://www.geeksforgeeks.org/deque-in-python/.
[37]. "collections - High-performance container datatypes." [Online]. Available: https://docs.python.org/2/library/collections.html.
[38]. "NVML API Reference Guide." https://docs.nvidia.com/deploy/nvml-api/index.html.
[39]. "plotly Dash." https://plotly.com/dash/.
[40]. M. Makai, "Green Unicorn (Gunicorn)." https://www.fullstackpython.com/green-unicorn-gunicorn.html#:~:text=Green%20Unicorn%20(Gunicorn),to%20run%20Python%20web%20applications.
[41]. N. Emirot, "Deploy a Flask app with Gunicorn and Docker." https://blog.nolanemirot.com/2016/03/11/deploy-a-flask-app-with-gunicorn-and-docker/.
[42]. I. Turner-Trauring, "Configuring Gunicorn for Docker." https://pythonspeed.com/articles/gunicorn-in-docker/.
[43]. "Design." [Online]. Available: https://docs.gunicorn.org/en/stable/design.html.