GPU Usage Reporting

Nicholas P. Cardo, Miguel Gila, Mark Klein
Swiss National Supercomputer Centre (CSCS)
HPC Operations
Lugano, Switzerland

***Abstract*—For systems with accelerators, such as Graphics Processing Units (GPUs), it is vital to understand their usage in solving scientific problems. Not only is this important to understand for current systems, but it also provides insight into future system needs. However, there are limitations and challenges that have prevented reliable statistic capturing, recording and reporting. The Swiss National Supercomputer Centre (CSCS) has developed a mechanism for capturing and storing GPU statistical information for each batch job. Additionally, a batch job summary report has been developed to display useful statistics about the job, including GPU utilization statistics. This paper will discuss the challenges that needed to be overcome along with the design and implementation of the solution.**

Keywords; HPC, GPU, Usage Reporting

I. INTRODUCTION

Understanding the utilization of computational resources is an important metric that is used for the sizing of next generation systems. However, with accelerators, this has been difficult to ascertain. Most accounting functionality will cover the host processor and the node itself. While it is possible for a user to obtain GPU usage information, this is not linked to the general accounting. A solution is needed that bridges the gap between statistical data from a GPU and the integrated accounting with workload managers. Once the information is integrated, reporting capabilities can be explored further. The simplest example is that with this data, it is possible to determine if a batch job simply used the node with the host processor or actually used the available GPU.

II. OBJECTIVES

In order to design a solution, the problem had to be broken down and clearly understood. What areas of work would be required to accomplish an integrated solution? To start with, the problem was simplified down to determining if a batch job utilized the available GPUs. With the end result defined, it was then possible to break down the problem of bridging the gap between the GPU and accounting. Three components of for an overall solution to providing GPU statistics were identified. These were:

1. Data Capture: the ability to capture statistics
2. Data Store: the ability to store statistics
3. Data Reports: the ability to report statistics

Now that the problem was understood, the next step was to define the boundary conditions for the solution. Several design objectives were established to guide the development efforts. These were:

1. Determine if a batch job utilized GPUs
2. Store statistics with job data accounting
3. Make the data available to users
4. Capability to report on all jobs

These objectives also became the success criteria for development.

III. CHALLENGES

Limitations in GPU counters restricted the amount of data that could be captured. This is an architectural limitation and changes from model to model. The CSCS flagship system Piz Daint utilizes the NVIDIA Tesla P100 GPU. The model of the GPU establishes what data is available and could be useful for data collection.

Piz Daint has more than 5,300 GPUs available for production computational service. For large scale applications, the challenge is how to capture the data and aggregate it across all used compute nodes for any given batch job in an efficient way. Computational resources are an expensive commodity that need to be maximally available for scientific computing.

Once the data has been captured, the next challenge was how to get the data off the system in a way that can be linked with job data. This would have been a simple problem if it weren't for the requirement of storing the data with the job's accounting data. Storing the data separately would have created the problem of how to link the data from data stores in potentially different locations and making it available to users and system reporting. The ideal solution is to store it all in one place.

With everything in place, the final challenge was how to expose this data to the users as well as for system level reporting. Users should be able to easily see the results without custom software development. The data also needed to be easily retrievable.

IV. DESIGN

CSCS utilizes the Slurm workload manager for controlling access to the computational resources. It was clear that the solution had to integrate seamlessly into the natural operation of the workload manager ideally without custom modifications. Having four clear objectives decomposed the problem into solvable components that in the end needed to work together to provide an integrated solution.

A. Satisfying Objective #1

Like most workload managers, Slurm includes the capability of executing a site defined prologue (executes before the job) as well as a site defined epilogue (executes after the job). Computing the GPU utilization difference between job startup and job completion yields the utilization for that job. Basic information regarding usage of the GPU was available directly from the GPU. The data could be reported as high-water marks and/or accumulated consumption of the resource. The following data was determined to be useful:

1. GPU seconds, averaged over the nodes
2. Maximum GPU second of a node
3. Maximum GPU memory of a node
4. Total GPU memory across all nodes

While not an exhaustive amount of data, it is sufficient to deliver on Objective #1: did a batch job utilize the GPU.

B. Satisfying Objective #2

Objective #2 requires that the newly captured data be stored in a manner that is integrated with a batch job's accounting data. The Slurm accounting records are fixed and modifying the database records was determined to be undesirable. However, carefully analysis of the data structure revealed that an existing variable text field could be utilized to hold the data. As previously identified, there are four data elements to store with only a single field to store them. A multi-variable storage format was needed that conversion tools were readily available. The solution was found in using JavaScript Object Notation (JSON) format. The design would allow for a JSON format string to be inserted into a job's record into the `AdminComment` field. All the data was now in a single storage location satisfying Objective #2.

C. Satisfying Objective #3

Objective #3 called for providing users access to the newly collected information. Since this was stored in the Slurm accounting data the solution came with the `sacct` command. This is the standard command for access accounting data for batch jobs. Command options allow for requesting the `AdminComment` field to be displayed. However, the information displayed would still be in JSON format. This created an additional requirement to have the tools available to decode the JSON formatted text. The provided tools needed to provide both a library and command-line method for parsing the text. Under CLE 6, the Jansson C library for JSON manipulation is already included enabling compiled codes to have a standard way to parse JSON. There are many command-line utilities available for parsing JSON formatted text. After trying several options, JQ was selected as the command-line tool to make available to users. With the data and tools available, Objective #3 is satisfied.

D. Satisfying Objective #4

In order to satisfy the reporting requirements of Objective #4, the information needed to be easily presented to the users without much work on their part. Furthermore, system level reporting capabilities are necessary. A batch job summary report could be presented to the user appended to the end of each batch job that contains the GPU usage information. Having the data included in the Slurm job accounting record would automatically expose the new data to system level reports. These reporting solutions satisfied the requirements of Objective #4. The design and implementation of system level reports would be developed separately.

V. Implementation

A custom plug-in was developed to perform the data capture and amortize it over the scale of the job. The functionality desired is very close in methodology to the way Cray's Resource Utilization Reporting (RUR) package worked. However, under native Slurm, RUR is not supported. What if the RUR functionality could be made to work and extended to handle the requirements of gathering GPU statistics? This is exactly the approach taken for the initial release.

A. JSON Format

JSON provides a standardized format with a large selection of tools for generation and processing. With this, several parameter and value pairs could be embedded within a single storage location. Five elements of data are embedded into the `AdminComment` field of the Slurm accounting record for a given batch job. The following is a sample JSON string:

```
{"gpustats":{
    "maxgpusecs": 146,
    "maxmem": 17034117120
    "gpupids": 1,
    "summem": 17034117120,
    "gpusecs": 146
}
```

Each field provides details about the usage of a GPU. These fields are:

- `maxgpusecs`: high water mark of seconds a GPU was in use across all used GPUs
- `maxmem`: high water mark of memory usage of a GPU across all used GPUs
- `gpupids`: GPU identifier
- `summem`: total accumulated memory usage across all used GPUs
- `gpussecs`: total accumulated seconds the GPUs were used across all used GPUs

Because JSON was chosen as the data format, it can easily be extended. Additional data elements can be inserted without the need to change what is already working. As long as the names of the existing fields are not changed others fields can be inserted. Also, location is not important as with JSON a field is requested by name. This is a very important factor as it enables an ease of growth. Over time as more data is identified, the string can be increased without impacting the existing production operation. Reporting mechanisms can then be adjusted asynchronously.

B. Adapting RUR

RUR utilizes a plugin architecture for capturing data. GPU data is obtained with the `gpustat_stage.py` plugin. A problem was found which was resulting in the gpu seconds being double reported. A minor update was made to correct this behavior.

```
    lines = re.split("\n", nvacct)
    for line in lines:
-     try:
+     if re.search("maxMemoryUsage=0",line):
+         continue
+     try:
```

This minor modification to first check memory consumption corrected the behavior.

Another modification was required for the `taskstats_stage.py` plugin in order to get RUR to work.

```
     if jobid:
-    if data.data['pjid'] != 0 and
         data.data['apid'] != 0:
+    if data.data['uid'] != 0 and
         data.data['apid'] != 0:
```

RUR was developed for systems running the Cray Application Level Placement Scheduler (ALPS). However, on Piz Daint, Slurm is configured in "native" mode which eliminates the need for ALPS. RUR would attempt to obtain the list of nodes for the job through ALPS which is not available in native mode. This was simple to overcome by simply switching to the environment variable `SLURM_JOB_NODELIST`.

The final part to RUR was to enable it to record the data into the Slurm accounting record for that job. By using the `file_output.py` plugin as a template, a new plugin was written specifically to record the Slurm data.

```
    jout={}
    line=line.replace('\r', '').replace('\n',
'').replace('[','').replace(']','').replace("
', ","'=").replace("'","").replace(",","")
    raw=dict(token.split('=') for token in
shlex.split(line))
    jout['gpustats']=dict((k,int(v)) for k,v
in raw.iteritems())
    jout=json.dumps(jout)
    command= "/usr/bin/mysql -h somesvr -u
someuser -p somepass somedb -e 'update
%s_job_table set admin_comment=\"%s\" where
id_job=%s and id_user=%s'" %
(cluster,jout.replace("\"",'\\"'),jobid,uid)
    subprocess.call(command,shell=True)
```

While this may appear complicated, it is simply cleaning up the JSON output and using MySQL to update the job's accounting record.

The main advantage of leveraging the technology in RUR is the embedded capability to accumulate the data from all nodes assigned to the batch job. RUR already had the capability to capture and accumulate data from a batch job's node list making it an ideal technology to leverage.

C. Batch Job Summary Report

A batch job summary report is produced in the epilog and appended to the jobs standard output file. However, at the time of generating the report, the job's standard output file is no longer accessible. A workaround to this is to write directly to the batch's job standard output file. The limitation that interactive jobs, jobs not created with `sbatch`, would not receive the output was an acceptable compromise. This limitation is created because there is no standard output file for interactive jobs.

Deciding what to automatically report back at the end of each job is very site specific and customized to the appropriate user community. At CSCS, the following elements of the report were determined to be required:

- Batch job life span
- Elapsed time
- Requested time
- Username
- Account name
- Slurm partition
- Number of nodes
- Consumed Energy
- GPU Usage Statistics, when on a hybrid node
- Scratch file system inode usage

This list was determined after an analysis of user tickets to answer the question of what could be provided that either would have eliminated a question, arrived at the true problem quicker, or eliminated additional work.

```
Batch Job Summary Report for Job "test1" (6802625) on daint
------------------------------------------------------------------------------------------------------------
             Submit            Eligible               Start                 End    Elapsed  Timelimit
------------------- ------------------- ------------------- ------------------- ---------- ----------
2018-04-12T06:58:40 2018-04-12T06:58:40 2018-04-12T06:58:41 2018-04-12T07:01:19   00:02:38   00:15:00
------------------------------------------------------------------------------------------------------------
Username    Account     Partition   NNodes   Energy
---------- ---------- ---------- ------ ------------
cardo       csstaff     debug            1   18.31K joules
------------------------------------------------------------
gpusecs  maxgpusecs        maxmem             summem
------- ---------- ------------ ------------
    146         146  17034117120       17034117120
------------------------------------------------------------
Scratch File System        Files       Quota
-------------------- ---------- ----------
/scratch/snx3000               2     1000000
```

From this report users can understand the basics about their job's resource consumption as well as clearly indicate their GPU usage. It is also true that a job that did not use an available GPU now becomes identifiable. By knowing the batch job's elapsed time and the `maxgpusecs`, an approximation can be made regarding how long a batch job used a GPU with respect to the job's overall run time.

The "Batch Job Summary Report" is a simple BASH script called from the `slurmctld` prolog that extracts the data from the Slurm accounting database and from the system.

VI. Early Results

After successful testing on the XC50 test and development system, Dom, the solution was moved to Piz Daint. Immediately after entering production, a large number of RUR timeouts were being recorded. Further investigation identified that data was not being recorded for all jobs. With the successfully working on Dom, it was likely related to scale

on Piz Daint. The problem was confirmed and the problem isolated to the open file limit on the tier-2 boot nodes for `rsyslogd`. Piz Daint has 7,134 compute nodes all trying to record data. The initial limit was set to 8,192 and causing some connections to timeout resulting in data not being recorded. The solution was to double the value to 16,384. With this change both problems disappeared; no more timeouts or missing data.

In the line of work in providing support for HPC systems, it is very rare to receive a compliment from a user. This in fact, was an unexpected pleasure to have reported through our trouble ticket system.

```
Dear CSCS

Since today I get Batch Job Summary Reports by
default at the end of my jobs.

They are very useful.  Thanks for enabling the
feature.  Well done.

Cheers
```

Appreciation from the user community is a clear sign of value provided to them.

VII. Next Steps

Given the flexibility of the design, other data elements will be evaluated to add to JSON text.

The development of system level reports is an ongoing effort. CSCS is currently working to define meaningful reports and/or metrics. Reporting could be performed down to the project level or a broader view from the system level. A conceptual idea for a report has been put together and work will progress to produce it for evaluation. This is likely to be an iterative process until meaningful reports are routinely produced.

The current solution is based on an adaptation of RUR. Considerations are being given to taking this to a completely independent solution. The end result should be a solution that in portable and can run on any system that employs GPUs.

The Data Center GPU Manager (DCGM) by NVIDIA continues to evolve. It may be possible to adopt `dcgm` into the solution.

VIII. Summary

This solution has been successfully running in full production since February 21, 2018. Data is continuously being collected for analysis. User level accessibility to the collected has been greatly simplified with the Batch Job Summary report that is automatically generated into their standard output file.

The initial deployment has been completed and the development of meaningful system level reports is now underway.

This endeavor has opened the door to future development opportunities including the possibility of delivering a system independent solution.

Acknowledgment

This work was supported by the Swiss National Supercomputing Centre (CSCS).

References

1. SchedMD, “Slurm Workload Manager,” [online]. Available: https://slurm.schedmd.com
2. NVIDIA Corporation, “NVIDIA,” [online]. Available: https://www.nvidia.com
3. NVIDIA Corporation, “Data Center GPU Manager,” [online]. Available: http://www.nvidia.com/object/data-center-gpu-manager.html
4. Cray Inc., “Cray,”, [online]. Available: https://www.cray.com
5. “JavaScript Object Notation (JSON),” [online]. Available: https://www.json.org
6. “Jansson,” [online]. Available: http://www.digip.org/jansson
7. “./jq,” [online]. Available: https://stedolan.github.io/jq
8. Oracle, “MySQL,” [online]. Available: https://www.mysql.com
9. Cray Inc., “*Resource Utilization Reporting*,” [online]. Available: https://pubs.cray.com/content/S-2393/CLE%206.0.UP05/xctm-series-system-administration-guide/resource-utilization-reporting
10. Cray Inc., “*XC Series Application Placemen Guide*,” [online]. Available: Cray Inc., https://pubs.cray.com/content/S-2496/CLE%206.0.UP01/xctm-series-user-application-placement-guide-cle-60up01