Jobstats: A Slurm-Compatible Job Monitoring Platform for CPU and GPU Clusters

Josko Plazonic
plazonic@princeton.edu
OIT Research Computing, Princeton University
Princeton, New Jersey, USA

Jonathan D. Halverson
halverson@princeton.edu
Princeton Institute for Computational Science and Engineering, Princeton University
Princeton, New Jersey, USA

Troy J. Comi
tcomi@princeton.edu
OIT Research Computing & Department of Chemical and Biological Engineering, Princeton University
Princeton, New Jersey, USA

ABSTRACT

Job monitoring on high-performance computing clusters is important for evaluating hardware performance, troubleshooting failed jobs, identifying inefficient jobs and more. The combination of the Prometheus monitoring framework and the Grafana visualization toolkit has proven successful in recent years. This work shows how four Prometheus exporters can be configured for a Slurm cluster to provide detailed job-level information on CPU/GPU efficiencies and CPU/GPU memory usage as well as node-level Network File System (NFS) statistics and cluster-level General Parallel File System (GPFS) activity. A novel approach was devised to efficiently store a summary of this data in the Slurm database for each completed job. The open-source job monitoring platform introduced here can be used for batch, interactive and Open OnDemand jobs. Several tools are presented that use the Prometheus and Slurm databases to create dashboards, utilization reports and alerts.

CCS CONCEPTS

• **Information systems → Computing platforms**.

KEYWORDS

Job Monitoring, Slurm, Prometheus, Grafana, GPUs, Alerts

ACM Reference Format:
Josko Plazonic, Jonathan D. Halverson, and Troy J. Comi. 2023. Jobstats: A Slurm-Compatible Job Monitoring Platform for CPU and GPU Clusters. In *Practice and Experience in Advanced Research Computing (PEARC '23), July 23–27, 2023, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3569951.3604396

1 INTRODUCTION

On a high-performance computing (HPC) cluster, users submit jobs to the workload manager which arranges for the work to be carried out on the compute nodes. The job scheduler typically provides only limited tools for monitoring various aspects of the running jobs. Due to the high cost of such clusters and the demand by users for high throughput, it is important to ensure that the resources are being used properly. Additionally, the complexity of HPC clusters leads to difficult-to-diagnose problems such as file system slowdowns, CPUs being throttled and failed jobs.

To address these matters, various job monitoring platforms have been introduced including Ganglia [9], XDMoD [12], TACC Stats [4], MAP [10, 11], LIKWID [22] and PIKA [2]. Some users of these platforms seek improvements such as greater support for GPU jobs, enhanced visualization tools, and better facilities for working with real-time job metrics. The combination of the Prometheus monitoring framework [23] and the Grafana visualization toolkit [24] overcome many of the shortcomings of previous job monitoring platforms. For example, Kunz et al. [8] used Prometheus and Grafana to detect a variety of anomalous jobs through simulation. The authors were able to automatically detect several types of errant jobs.

In the present work, we introduce the Jobstats job monitoring platform. The platform is based on Prometheus, Grafana and the Slurm job scheduler [7]. Jobstats has proven to be of great value at our institution which has a data center with 100,000 CPU-cores and 500 GPUs. We discuss Prometheus and how it interacts with the various components of the Jobstats platform in Section 2. Tools that build on the platform are described in Section 3.

2 DESIGN OF THE JOBSTATS PLATFORM

2.1 Overview

The Jobstats platform is built on the Prometheus monitoring framework [23] which provides a fast and efficient time-series database. Such a database is a requirement for large clusters where the number of collected data points can be exorbitant. Job and node statistics are exposed by four different Prometheus exporters or programs that collect local statistics on a node and make them available for the Prometheus server to collect. On the compute nodes, there are up to three exporters: a standard node exporter for monitoring generic statistics (e.g., CPU frequencies, NFS and local disk I/O statistics), CPU utilization and CPU memory usage data of individual jobs via a cgroup exporter and, optionally, GPU job statistics via a modified NVIDIA GPU exporter. An optional fourth exporter tracks Spectrum Scale/GPFS filesystem use, only one instance per cluster, usually on a central GPFS server and not on any of the compute nodes (as a single instance exposes data for all members of the GPFS cluster).

A central Prometheus server collects data from all of these exporters every N seconds and stores it in its database with a retention period of several months. We find that a choice of $N = 30$ seconds

PEARC '23, July 23–27, 2023, Portland, OR, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9985-2/23/07.
https://doi.org/10.1145/3569951.3604396

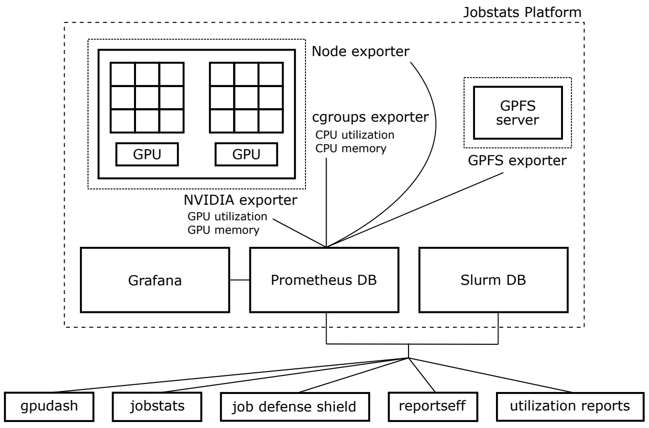

Figure 1: A schematic diagram of the components of the Jobstats platform and the external tools. A compute node with two sockets is shown in the upper left. The dotted line around the node indicates the three node-level exporters, namely, Node, cgroups and NVIDIA. A GPFS server is shown in the upper right with its cluster-level GPFS exporter. The exporters serve to make data availalble to the Prometheus database. Users interact with the Prometheus data via Grafana and the external tools (e.g., `gpudash`, `jobstats`).

yields sufficient data while only introducing a very small performance overhead penalty. Kunz et al. [8] used a value of 5 seconds. For a quick overview of job statistics and for long-term retention, a summary of individual job statistics is generated at job completion and stored in the Slurm database in the `AdminComment` field. This is done by a `slurmctld` epilog script that runs at job completion, combined with a clean-up script that checks every 5 minutes if there are jobs that did not get the summary and then generates it. The format of the `AdminComment` data is described below.

The `jobstats` command is used by users to display job efficiency summaries. For completed jobs this is done by retrieving data from the `AdminComment` field, while for actively running jobs it is done by querying the Prometheus database directly. The output includes per-node and overall CPU utilization and CPU memory usage as well as the analogous quantities for GPU jobs.

Detailed plots showing the time history of various quantities are available on the Grafana server(s), usually via an Open OnDemand helper script that creates URLs with the appropriate jobids, start times and end times.

Below is an outline of the steps that need to be taken to setup the Jobstats platform for a Slurm cluster:

(1) Switch to cgroup based job accounting from Linux process accounting
(2) Setup the exporters: cgroup, node, GPU (on the nodes) and, optionally, GPFS (centrally)
(3) Setup the prolog.d and epilog.d scripts on the GPU nodes
(4) Setup the Prometheus server and configure it to scrape data from the compute nodes and all configured exporters
(5) Setup the `slurmctldepilog.sh` script for long-term job summary retention
(6) Lastly, configure Grafana and Open OnDemand.

Each of the steps above are discussed in detail below. For additional instructions, URLs and scripts, see the Jobstats GitHub repository [17].

2.2 CPU Utilization and CPU Memory

Slurm has to be configured to track job accounting data via the `cgroup` plugin. This requires the following line in `slurm.conf`:

```
JobAcctGatherType=jobacct_gather/cgroup
```

The above is in addition to the other usual `cgroup` related plugins/settings:

```
ProctrackType=proctrack/cgroup
TaskPlugin=affinity,cgroup
```

Slurm will then create two top-level `cgroup` directories for each job, one for CPU utilization and one for CPU memory [17]. Within each directory there will be subdirectories: `step_extern`, `step_batch`, `step_0`, `step_1`, and so on. Within these directories one finds `task_0`, `task_1`, and so on. These cgroups are scraped by a `cgroup` exporter [14]. Table 1 lists all of the collected fields.

The `cgroup` exporter used here is based on Ref. [3] with additional parsing of the jobid, steps, tasks and UID number. This produces an output that resembles (e.g., for system seconds):

```
cgroup_cpu_system_seconds{jobid="247463",
                          step="batch",task="0"}
160.92
```

Note that the UID of the owning user is stored as a gauge in cgroup_uid:

```
cgroup_uid{jobid="247463"}
334987
```

This is because accounting is job-oriented and having a UID of the user as a label would needlessly increase the cardinality of the data in Prometheus. All other fields are alike with jobid, step and task labels.

The totals for a job have an empty step and task, for example:

```
cgroup_cpu_user_seconds{jobid="247463",
                        step="",task=""}
202435.71
```

This is due to the organization of the `cgroup` hierarchy. Consider the directory:

```
/sys/fs/cgroup/cpu,cpuacct/slurm/uid_334987
```

Within this directory, one finds the following subdirectories:

```
job_247463/cpuacct.usage_user
job_247463/step_extern/cpuacct.usage_user
job_247463/step_extern/task_0/cpuacct.usage_user
```

This is the data most often retrieved and parsed for overall job efficiency which is why by default the cgroup_exporter does not parse step or task data. To collect all of it, add the `--collect.fullslurm` option. We run the cgroup_exporter with these options:

```
/usr/sbin/cgroup_exporter --config.paths /slurm \
                          --collect.fullslurm
```

The `--config.paths /slurm` has to match the path used by Slurm under the top `cgroup` directory. This is usually a path that is something like `/sys/fs/cgroup/memory/slurm`.

Table 1: cgroup metrics made available by the cgroups exporter.

Name	Description	Type
cgroup_cpu_system_seconds	Cumulative CPU system seconds for jobid	gauge
cgroup_cpu_total_seconds	Cumulative CPU total seconds for jobid	gauge
cgroup_cpu_user_seconds	Cumulative CPU user seconds for jobid	gauge
cgroup_cpus	Number of CPUs in the jobid	gauge
cgroup_memory_cache_bytes	Memory cache used in bytes	gauge
cgroup_memory_fail_count	Memory fail count	gauge
cgroup_memory_rss_bytes	Memory RSS used in bytes	gauge
cgroup_memory_total_bytes	Memory total given to jobid in bytes	gauge
cgroup_memory_used_bytes	Memory used in bytes	gauge
cgroup_memsw_fail_count	Swap fail count	gauge
cgroup_memsw_total_bytes	Swap total given to jobid in bytes	gauge
cgroup_memsw_used_bytes	Swap used in bytes	gauge
cgroup_uid	UID number of user running this job	gauge

2.3 GPU Job Statistics

GPU metrics (currently only NVIDIA) are collected by our exporter [19] which was based on Ref. [1]. The main local changes were to add the handling of Multi-Instance GPUs (MIG) and two additional gauge metrics: nvidia_gpu_jobId and nvidia_gpu_jobUid. Table 2 lists all of the collected GPU fields. Note that the approach described here is not appropriate for clusters that allow for GPU sharing (e.g., sharding). In Section 3, we demonstrate how the GPU metrics stored in the Prometheus database can be queried by tools that generate dashboards and utilization reports.

2.4 Node Specific Statistics

A standard node_exporter runs on every node. This allows us to obtain other basic node metrics such as total memory available, memory in use, CPU frequencies, NFS statistics, Infiniband statistics and many other potentially useful data points. Spectrum Scale/GPFS statistics are collected with a custom Python based exporter [16].

2.5 Generating Job Summaries

Job summaries, as described above, are generated and stored in the Slurm database at the end of each job by using a `slurmctld` epilog script. For example, in `slurm.conf`:

```
EpilogSlurmctld=/usr/local/sbin/slurmctldepilog.sh
```

The script is available in the Jobstats GitHub repository [15].

For storage efficiency and convenience, the JSON job summary data is gzipped and base64 encoded before being stored in the `AdminComment` field of the Slurm database. The impact on the database size due to this depends on job sizes. On our clusters, for small jobs the `AdminComment` field tends to average under 50 characters per entry with a maximum under 1500 while for large jobs the maximum length is around 5000.

2.6 Grafana

The four exporters lead to a wealth of data in the Prometheus database. To visualize this data, the Grafana visualization toolkit [24] is used. The following *job-level* metrics are available in both Grafana and the `jobstats` command:

- CPU Utilization
- CPU Memory Utilization
- GPU Utilization
- GPU Memory Utilization

The following additional *job-level* metrics are exposed only in Grafana:

- GPU Temperature
- GPU Power Usage

Finally, the following additional *node-level* metrics are exposed only in Grafana:

- CPU Percentage Utilization
- Total Memory Utilization
- Average CPU Frequency Over All CPUs
- NFS Statistics
- Local Disc R/W
- GPFS Bandwidth Statistics
- Local Disc IOPS
- GPFS Operations per Second Statistics
- Infiniband Throughput
- Infiniband Packet Rate
- Infiniband Errors

Eleven of the seventeen metrics above are node-level. This means that if multiple jobs are running on the node then it will not be possible to disentangle the data. To use these metrics to troubleshoot jobs, the job should allocate the entire node.

The complete Grafana interface for the Jobstats platform is composed of plots of the time history of the seventeen quantities above. An example of the Grafana dashboard and the needed code are available in the Jobstats GitHub repository [17]. This graphical interface is used for detailed investigations such as troubleshooting failed jobs, identifying jobs with CPU memory leaks, intermittent GPU usage, load imbalance, and for understanding the anomalous behavior of system hardware.

While the Grafana interface is an essential component of the Jobstats platform, for quick inspections of job behavior, the `jobstats` command is used. This tool and four others are discussed in Section 3.

Table 2: GPU metrics made available by the NVIDIA exporter.

Name	Description	Type
nvidia_gpu_duty_cycle	GPU utilization	gauge
nvidia_gpu_memory_total_bytes	Total memory of the GPU device in bytes	gauge
nvidia_gpu_memory_used_bytes	Memory used by the GPU device in bytes	gauge
nvidia_gpu_num_devices	Number of GPU devices	gauge
nvidia_gpu_power_usage_milliwatts	Power usage of the GPU device in milliwatts	gauge
nvidia_gpu_temperature_celsius	Temperature of the GPU device in Celsius	gauge
nvidia_gpu_jobId	JobId number of a job currently using this GPU as reported by Slurm	gauge
nvidia_gpu_jobUid	UID number of user running jobs on this GPU	gauge

3 TOOLS OF THE JOBSTATS PLATFORM

The Prometheus and Slurm databases provide a rich dataset that can be harnessed by special-purpose tools:

- `jobstats`: A command for generating a detailed Slurm efficiency report for a given job.
- `job defense shield`: A tool for sending automated email alerts to users with underperforming jobs.
- gpudash: A command that generates a dashboard showing the utilization of each GPU on the cluster.
- `reportseff`: A command for displaying a simple Slurm efficiency report for several jobs at once.
- `utilization reports`: A tool for sending detailed usage reports to users and group leaders by email.

More details about each of these tools are provided below.

3.1 jobstats

The `jobstats` command provides users with a Slurm job efficiency report. For completed jobs, the data is taken from a call to `sacct` with several fields including `AdminComment`. For running jobs, the Prometheus database must be queried using the following:

```
max_over_time(cgroup_memory_total_bytes{...}[...])
max_over_time(cgroup_memory_rss_bytes{...}[...])
max_over_time(cgroup_cpu_total_seconds{...}[...])
max_over_time(cgroup_cpus{...}[...])
```

See the GitHub repository [17] for the additional queries needed for actively running GPU jobs.

The `jobstats` command requires a jobid:

```
$ jobstats 247463
```

An example of the `jobstats` output is available at https://github.com/PrincetonUniversity/jobstats. The first part of the output displays job metadata such as the username, account, partition, cluster, number of CPU-cores, start time and so on. The second part uses a text-based meter to indicate the overall CPU/GPU utilization and CPU/GPU memory usage. Detailed information is provided in the third part of the output which includes per-node, per-CPU and per-GPU values for the utilization and memory usage. The final panel includes useful notes for the user based on the job metadata and the CPU/GPU efficiencies and memory used. Such notes have proven useful elsewhere [6]. Below is an example note from a job that used 6 CPU-cores and over-allocated CPU memory:

```
  * This job only used 15% of the 100GB of total
allocated CPU memory. Please consider allocating less
memory by using the Slurm directive --mem-per-cpu=3G or
--mem=18G. This will reduce your queue times and make
the resources available to other users. For more info:
https://researchcomputing.princeton.edu/memory
```

For our institution, there are currently more than twenty possible notes. They cover such issues as low CPU or GPU utilization, over-allocating CPU memory, using excessive run time limits, allocating more nodes than necessary (i.e., job fragmentation), running serial codes with multiple CPU-cores, and running jobs in the test queue. Color and bold font is used throughout the report to draw the user's attention to key pieces of information. If a job runs for less than twice the sampling period of the Prometheus exporters (60 seconds) then the `seff` command is used in place of `jobstats`.

Importantly, the `jobstats` command is also used for Slurm efficiency reports that are sent by email after a job completes. This is done by changing the `MailProg` setting in `slurm.conf`. For details, see Appendix A.2.

The installation requirements for `jobstats` are Python 3.6+ and version 2.20+ of the Python `Requests` package. Version 1.17+ of the Python `blessed` package [21] is optional. If `blessed` is available then it will be used for coloring and styling text. The Python code and instructions are available in the GitHub repository [17].

3.2 Job Defense Shield

High-performance computing clusters often serve a large number of users who posses a range of knowledge and skills. This leads to individuals misusing the resources due to mistakes, misunderstandings, expediency, and related issues. To combat jobs that waste or misuse the resources, a battery of alerts can be configured. While such alerts can be configured in Prometheus [23], the most flexible and powerful solution is external software.

The `job defense shield` is a Python code for sending automated email alerts to users and for creating reports for system administrators. As discussed above, summary statistics for each completed job are stored in a compressed format in the `AdminComment` field in the Slurm database. The software described here works by calling the Slurm `sacct` command while requesting several fields including the `AdminComment` field. The `sacct` output is stored in a pandas dataframe for processing.

The `job denfense shield` provides email alerts for the following:

- actively running jobs where a CPU or GPU has zero utilization for longer than a threshold time value

- users in the top N by usage over some time window with low CPU or GPU utilization
- jobs that could have been run on a less powerful GPU (e.g., an NVIDIA MIG GPU versus H100)
- jobs with excessive run time limits
- jobs that request too many nodes (e.g., 1 CPU-core per node over 10 nodes)
- jobs that run a serial code while allocating more than 1 CPU-core
- jobs that request much more than the default CPU memory but do not use it.

The Python code is written using object-oriented programming techniques which makes it easy to create new alerts.

The `job denfense shield` has a `check` mode that shows on which days a given user received an alert of a given type. Users that appear to be ignoring the email alerts can be contacted directly. Emails to users are most effective when sent sparingly. For this reason, there is a command-line parameter to specify the amount of time that must pass before the user can receive another email of the same nature.

The example below shows how the script is called to notify users in the top N by usage with low CPU or GPU efficiencies over the last week:

```
$ ./job_defense_shield.py --low-xpu-efficiencies \
                          --days=7 --email
```

The default thresholds are 60% and 15% for CPU and GPU utilization, respectively, and $N = 15$.

The installation requirements for the `job defense shield` are Python 3.6+ and version 1.2+ of the Python pandas package [13]. The `jobstats` command is also required if one wants to examine actively running jobs such as when looking for jobs with zero GPU utilization. The Python code, example alerts and emails, and instructions are available at https://github.com/PrincetonUniversity/job_defense_shield.

3.3 gpudash

The `gpudash` command generates a text-based dashboard of the GPU utilization across a cluster in the form of a 2-dimensional grid. Each cell displays the utilization from 0-100% along with the username associated with each allocated GPU. Cells are colored according to their utilization values making it easy to identify jobs with low or high GPU utilization. The `gpudash` command can also be used to check for available GPUs.

By default, the dashboard has seven columns and a number of rows equal to the number of GPUs on the cluster. Each column is evenly spaced in time by N minutes. We find a good choice is $N = 10$ minutes which leads to data being shown over an hour. The `cron` utility can be used to achieve this. The rows are labeled by the node name and the GPU index while the columns are labeled by time.

The `gpudash` command works by making the three queries to the Prometheus server every N minutes [5]. A Python script is used to extract the information from the three generated JSON files and append this data to the files read by `gpudash`. The UID for each user is matched with its corresponding username. The jobid is not required but it can be useful for troubleshooting.

Nodes that are down, or in a state which makes them unavailable, are not shown in the visualization. Special labels can be added to mark reserved nodes or special-purpose nodes. The installation requirements for `gpudash` are Python 3.6+ and version 1.17+ of the Python `blessed` package [21] which is used for creating colored text and backgrounds. The Python code and instructions are available at https://github.com/PrincetonUniversity/gpudash.

3.4 reportseff

The `reportseff` utility wraps `sacct` to provide a cleaner user experience when interrogating Slurm job efficiency values for multiple jobs. In addition to multiple jobids, `reportseff` accepts Slurm output files as arguments and parses the jobid from the filename. Some `sacct` options are further wrapped or extended to simplify common operations. The output is a table with entries colored based on high/low utilization values. The columns and formatting of the table can be customized based on command line options.

A limit to the previous tools is that they provide information on a single job at a time in great detail. Another common use case is to summarize job efficiency for multiple jobs to gain a better idea of the overall utilization. Summarized reporting is especially useful with array jobs and workflow managers which interface with Slurm. In these cases, running `seff` or `jobstats` becomes burdensome. `reportseff` accepts jobs as jobids, Slurm output files, and directories containing Slurm output files:

```
# get information on jobs 123 and 124
$ reportseff 123 124
# get information on jobs 123 to 133
$ reportseff {123..133}
# check output files starting with jobname
$ reportseff jobname*
# look for output files in the slurm_out directory
$ reportseff slurm_out/
```

The ability to link Slurm outputs with job status simplifies locating problematic jobs and cleaning up their outputs.

The `reportseff` utility extends some of the `sacct` options. The start and end time can accept any format accepted by `sacct`, as well as a custom format, specified as a comma separated list of key/value pairs. For example:

```
$ reportseff --since now-27hours  # equivalent to
$ reportseff --since d=1,h=3  # 1 day, 3 hours
```

Filtering by job state is expanded with `reportseff` to specify states to exclude. This filtering combined with accepting output files helps in cleaning up failed output jobs:

```
$ reportseff \
  --not-state CD \  # not completed
  --since d=1 \  # today
  --format=jobid \  # just get file name
  my_failing_job* \  # only from these outputs
  | xargs grep "output:"
```

The last piece of the pipeline above find lines with the output directive to examine or delete. The format option can accept a comma-separated list of column names or additional columns can be appended to the default values. Appending prevents the need to add in the same, default columns on every invocation.

While the above features are available for any Slurm system, when Jobstats information is present in the `AdminComment`, the multi-node resource utilization is updated with the more accurate Jobstats values and GPU utilization is also provided. This additional information is controlled with the `--node` and `--node-and-gpu` options.

A sample workflow with `reportseff` is to run a series of jobs, each producing an output file. Run `reportseff` on the output directory to determine the utilization and state of each job. Jobs with low utilization or failure can be examined more closely by copy/pasting the Slurm output filename from the first column. Outputs from failed jobs can be cleaned automatically with a version of the command piping above. Combining with `watch` and aliases can create powerful monitoring for users:

```
# monitor the current directory every 5 minutes
$ watch -cn 300 reportseff --modified-sort
# monitor the user's efficiency every 10 minutes
$ watch -cn 600 reportseff --user $USER \
        --modified-sort --format=+jobname
```

The installation requirements for `reportseff` are Python 3.7+ and version 6.7+ of the Python `click` package which is used for creating colored text and command line parsing. The Python code and instructions are available at https://github.com/troycomi/reportseff.

3.5 Utilization Reports

Users can receive an email utilization report upon completion of each job via Slurm directives. Because some users decide not to receive these emails, it is important to periodically send a comprehensive utilization report to each user. As discussed above, summary statistics for each completed job are stored in a compressed format in the `AdminComment` field in the Slurm database. The software described here works by calling `sacct` while requesting several fields including `AdminComment`. The `sacct` output is stored in a pandas dataframe for processing.

Each user that ran at least one Slurm job in the specified time interval will receive a report when the software is run. The first part of the report is a table that indicates the overall usage for each cluster. Each row provides the CPU-hours, GPU-hours, number of jobs, and Slurm account(s) and partition(s) that were used by the user.

The second part of the report is a detailed table showing for each partition of each cluster the CPU-hours, CPU-rank, CPU-eff, GPU-hours, GPU-rank, GPU-eff and number of jobs. The CPU-rank or GPU-rank indicates the user's usage relative to the other users on the given partition of the cluster. CPU-eff (or GPU-eff) is the overall CPU (or GPU) efficiency which varies from 0-100%. A responsible user will take action when seeing that their rank is high while their efficiency is low. The email report also provides a definition for each reported quantity. The software could be extended by adding queue hours and data storage information to the tables.

The default mode of the software is to send user reports. It can also be used to send reports to those that are responsible for the users such as the principal investigator. This is the so-called `sponsors` mode. The example below shows how the script is called to generate user reports over the past month which are sent by email:

```
$ python utilization_reports.py --report-type=users \
                                 --months=1 --email
```

We find a good choice is to send the report once per month. The installation requirements for the software are Python 3.6+ and version 1.2+ of the pandas package [13]. The Python code, example reports, and instructions are available at https://github.com/PrincetonUniversity/monthly_sponsor_reports.

4 SUMMARY

The Jobstats platform is built on Prometheus, Grafana and Slurm. The speed and efficiency of the time-series database provided by Prometheus is central to our design. The multiple exporters produce a rich dataset which is harnessed by external tools such as `jobstats`. A standard server is sufficient to support Prometheus. The Jobstats platform has proven successful at our institution which has 100,000 CPU-cores and 500 GPUs using an exporter sampling period of 30 seconds. While setting up the platform requires touching numerous files, the procedure for doing so is well-documented and the benefits of using the Jobstats platform are numerous. The platform is particularly relevant to institutions with GPU clusters. The custom notes that appear at the bottom of the `jobstats` output have proven to be very useful in guiding users. There are plans to extend the platform to provide information on data storage.

Can the platform be configured for job schedulers other than Slurm such as PBS? The most difficult piece to adjust is likely to be the cgroups-based process accounting. The other exporters should be easier to modify for other schedulers. Presumably there is something analogous to using the `AdminComment` field for the target scheduler.

Please direct all questions concerning this work to Princeton Research Computing at cses@princeton.edu or post an issue on the Jobstats GitHub repository [17].

ACKNOWLEDGMENTS

Discussions with William Wichser helped to shape the design of the Jobstats platform. The authors are grateful to Galen Collier, Kevin Abbey and Asya Dvorkin for reviewing this work. Carolina Roe-Raymond and Kevin Abbey contributed to the custom notes generated by the `jobstats` command.

REFERENCES

[1] Rohit Agarwal. 2018. NVIDIA GPU Prometheus Exporter. Retrieved March 3, 2023 from https://github.com/mindprince/nvidia_gpu_prometheus_exporter

[2] Robert Dietrich, Frank Winkler, Andreas Knüpfer, and Wolfgang Nagel. 2020. PIKA: Center-Wide and Job-Aware Cluster Monitoring. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, New York, NY, 424–432. https://doi.org/10.1109/CLUSTER49012.2020.00061

[3] Trey Dock. 2022. cgroup Prometheus exporter. Retrieved March 3, 2023 from https://github.com/treydock/cgroup_exporter

[4] Todd Evans, William L. Barth, James C. Browne, Robert L. DeLeon, Thomas R. Furlani, Steven M. Gallo, Matthew D. Jones, and Abani K. Patra. 2014. Comprehensive Resource Use Monitoring for HPC Systems with TACC Stats. In *2014 First International Workshop on HPC User Support Tools (HUST)*. IEEE, New York, NY, 13–21. https://doi.org/10.1109/HUST.2014.7

[5] Jonathan Halverson. 2023. gpudash. Retrieved June 15, 2023 from https://github.com/PrincetonUniversity/gpudash

[6] Petar Jager. 2023. goslmailer. Retrieved June 14, 2023 from https://github.com/CLIP-HPC/goslmailer

[7] Morris A. Jette, Andy B. Yoo, and Mark Grondona. 2003. Slurm: Simple Linux Utility for Resource Management. In *Lecture Notes in Computer Science (Job Scheduling Strategies for Parallel Processing, Vol. 2862)*. Springer-Verlag, Berlin, Germany, 44–60.

[8] Pascal Kunz. 2022. *HPC Job-Monitoring with Slurm, Prometheus and Grafana.* Bachelor Thesis. University of Basel, Spiegelgasse 1, 4051 Basel, Switzerland.
[9] Matthew L. Massie, Brent N. Chun, and David E. Culler. 2004. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Comput.* 30, 7 (2004), 817–840.
[10] Ashish Pal and Preeti Malakar. 2020. MAP: A Visual Analytics System for Job Monitoring and Analysis. In *2020 IEEE International Conference on Cluster Computing (CLUSTER).* IEEE, New York, NY, 442–448. https://doi.org/10.1109/CLUSTER49012.2020.00063
[11] Ashish Pal and Preeti Malakar. 2021. An Integrated Job Monitor, Analyzer and Predictor. In *2021 IEEE International Conference on Cluster Computing (CLUSTER).* IEEE, New York, NY, 609–617. https://doi.org/10.1109/Cluster48925.2021.00091
[12] J. T. Palmer, S. M. Gallo, T. R. Furlani, M. D. Jones, R. L. DeLeon, J. P. White, N. Simakov, A. K. Patra, J. Sperhac, T. Yearke, R. Rathsam, M. Innus, C. D. Cornelius, J. C. Browne, W. L. Barth, and R. T. Evans. 2015. Open XDMoD: A Tool for the Comprehensive Management of High Performance Computing Resources. *Computing in Science Engineering* 17, 4 (2015), 52–62.
[13] The pandas development team. 2020. *pandas-dev/pandas: Pandas.* Zenodo. https://doi.org/10.5281/zenodo.3509134
[14] Josko Plazonic. 2023. cgroup Prometheus Exporter. Retrieved March 3, 2023 from https://github.com/plazonic/cgroup_exporter
[15] Josko Plazonic. 2023. Generating Job Summaries with slurmctldepilog.sh. https://github.com/PrincetonUniversity/jobstats/blob/main/slurm.
[16] Josko Plazonic. 2023. GPFS Prometheus Exporter. Retrieved March 3, 2023 from https://github.com/plazonic/gpfs-exporter
[17] Josko Plazonic. 2023. Jobstats Job Monitoring Platform. Retrieved March 3, 2023 from https://github.com/PrincetonUniversity/jobstats
[18] Josko Plazonic. 2023. Jobstats Mail. Retrieved March 3, 2023 from https://github.com/PrincetonUniversity/jobstats/blob/main/slurm/jobstats_mail.sh
[19] Josko Plazonic. 2023. NVIDIA GPU Prometheus Exporter. Retrieved March 3, 2023 from https://github.com/plazonic/nvidia_gpu_prometheus_exporter
[20] Josko Plazonic. 2023. Prolog and Epilog Scripts. Retrieved March 3, 2023 from https://github.com/PrincetonUniversity/jobstats/tree/main/slurm
[21] Erik Rose, Jeff Quast, and Avram Lubkin. 2023. Python blessed Package. Retrieved March 3, 2023 from https://blessed.readthedocs.io/en/latest/
[22] Thomas Röhl, Jan Eitzinger, Georg Hager, and Gerhard Wellein. 2017. LIKWID Monitoring Stack: A Flexible Framework Enabling Job Specific Performance Monitoring for the Masses. In *2017 IEEE International Conference on Cluster Computing (CLUSTER).* IEEE, New York, NY, 781–784. https://doi.org/10.1109/CLUSTER.2017.115
[23] Julius Volz and Björn Rabenstein. 2023. Prometheus. Retrieved March 3, 2023 from https://prometheus.io/docs/introduction/overview/
[24] Torkel Ödegaard. 2023. Grafana. Retrieved March 3, 2023 from https://grafana.com

A PROMETHEUS DETAILS

A.1 GPU Jobs

For efficiency and simplicity, JobId and jobUid are collected from files in `/run/gpustat/0` (for GPU 0), `/run/gpustat/1` (for GPU 1), and so on. For example:

```
$ cat /run/gpustat/0
247609 223456
```

In the above, the first number is the jobid and the second is the UID number for that job's owning user. These are created with Slurm prolog.d and epilog.d scripts that can be found in the Jobstats GitHub repository [20].

A.2 Procedure for Modifying User Email Reports

To generate email reports using `jobstats` after a job finishes, the following line is needed in `slurm.conf`:

```
MailProg=/usr/local/bin/jobstats_mail.sh
```

Here are the key lines in the `jobstats_mail.sh` script:

```
SEFF=/usr/local/bin/jobstats --no-color
$SEFF $jobid | $MAIL -s "$subject" $recipient
```

One also needs to set the `content-type` to `text/html` so that the email uses a fixed-width font. The full script is available in the Jobstats GitHub repository [18].

Received 3 March 2023; revised X XXXXX 2023; accepted X XXXXX 2023