Learning from Five-year Resource-Utilization Data of Titan System

Feiyi Wang*, Sarp Oral†, Satyabrata Sen ‡ and Neena Imam§

Oak Ridge National Laboratory

Email: *fwang2@ornl.gov, †oralhs@ornl.gov, ‡sens@ornl.gov, §imamn@ornl.gov

***Abstract*—Titan was the flagship supercomputer at the Oak Ridge Leadership Computing Facility (OLCF). It was deployed in late 2012, became the fastest supercomputer in the world and was retired on August 2, 2019. With Titan's mission complete, this paper provides a first-order examination of the usage of its critical resources (CPU, Memory, GPU, and I/O) over a five-year production period (2015-2019). In particular, we show quantitatively that the majority of CPU time was spent on the large-scale jobs, which is consistent with the policy of driving ground-breaking science through leadership computing. We also corroborate the general observation of the low CPU-memory usage with 95% jobs utilizing only 15% or less available memory. Additionally, we correlate the increase of total job submissions and the decrease of GPU-enabled jobs during 2016 with the GPU reliability issue which impacted the large-scale runs. We further show the surprising read/write ratio over the five-year period, which contradicts the general mindset of the large-scale simulation machines being "write-heavy". This understanding will have potential impact on how we design our next-generation large-scale storage systems. We believe that our analyses and findings are going to be of great interest to the high-performance computing (HPC) community at large.**

I. Introduction

Titan served as the Oak Ridge Leadership Computing Facility (OLCF)'s flagship computing platform from 2012 until its decommission on August 2019. It was ranked No.1 on the TOP500 list at the time of its deployment and was still No.12 when it was retired. U.S. Department of Energy funded the Titan system with the mission to provide a leadership compute platform to a limited number of high-impact, grand-challenge scale projects. Architecturally, Titan was a hybrid-architecture Cray XK7 system that had 18,688 nodes connected with a proprietary Gemini 3D torus network. Each node consisted of a 16-core AMD Interlagos CPU and an NVidia K20X Kepler GPU, as well as 32 GiB of DDR3 system memory and 6 GiB of GDDR5 GPU memory. Further details on Titan can be found in [2], [3].

During its production lifetime, Titan provided more than 26 billion core hours of computing time to scientists. Throughout this period, an extensive operational dataset - called the Resource Utilization Report (RUR) - was collected from the Titan system. This RUR data, on one hand, is extremely coarse-grained: in order to avoid any noticeable disturbance to the production jobs, only a small amount of data could be collected and stored. On the other hand, the RUR data is extremely comprehensive in the sense that it provides a log record for every job submitted to Titan from April 2015 to July 2019. These records provide a unique window into operational resource usage at an extreme scale and over a long term. Our work presents a first-order analysis of the RUR data from Titan; however, it is also worth noting that this work by no means represents an exhaustive look and analysis of the entire RUR data. We expect to provide further analyses on this data in future.

This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan)

II. Resource Utilization Report

The Resource Utilization Report (RUR) is a Cray-developed, resource-usage data collection and reporting system that strives to be systematic, extensive, and lightweight. There are two key ideas behind the design. The first is the concept of pre-job and post-job wrappers. A pre-job wrapper is invoked before a job run. Its primary responsibility is to take a snapshot of system state, which is to be used as a point of reference later. The post-job wrapper takes another snapshot after the job exit (failed or successful), and performs both tally and size-reduction on the data that needs to be recorded. The second key design idea is the plugin infrastructure. As there are many system components of interest to be tracked and monitored, it is not possible to cover them all. Thus, the RUR data collection is essentially a collection of individual scripts that are called based on site-specific needs. Since the pre- and post-wrapper only run at the beginning and ending stages of each job session (i.e., not while the actual job is running), there is no performance disturbance to the job itself. An earlier design paper by Andrew Barry [1] provided additional information on related work and design philosophy.

For each job, we aggregate the per-component raw data into a single json-formatted output, as shown in the following snippet:

```
{
  # from Cray job scheduler (alps)
  "alps_width": [2048],
  "alps_depth": [1],
  "aprun_id": 9698386, ...
  # from taskstats plugin
```

TABLE I
TITAN RUR DATA

Year	Raw data size	# of job submissions	# of failed jobs	% of failed jobs
2015	1.5 GB	1,529,972	292,699	19%
2016	4.3 GB	4,745,305	691,094	14%
2017	2.8 GB	2,814,838	328,187	11%
2018	2.2 GB	2,370,860	250,773	10%
2019	1.3 GB	1,520,971	104,173	6%

```
  "taskstats":{
    "max_rss":31196,"rchar":40948218,
    "stime":6308596000, ...
  },
  # from gpu plugin
  "gpustat":{"gpupids":0,"gpusecs":0,
  "maxgpusecs":0,"maxmem":0,"summem":0},
  ...
}
```

For a list of supported plugins, metrics and their definitions, please refer to Cray's official document [4]. Jim Rogers discussed particular issues and challenges in analyzing GPU usage with Cray's RUR on Titan [6].

Table I summarizes the collected RUR data. On disk, the raw data files are organized in three-level structures: by *year*, *month*, and *day*. For the ease of analysis, we coalesced all the files into a single raw file on a yearly basis. The column **raw data size** shows the size of raw data files for every year. By carefully controlling what to collect and how much to report, we can minimize the post-scripts' runtime and make storage-needs more manageable. The **number of job submissions** column shows the number all job submitted for every year. And the fourth column is a tally of **failed job** submissions based on the RUR alps_exit status field - it is a successful run only if job return status is zero.

There are two related trends observed from this data. First, we notice that the overall failed job rate steadily declined over the five-year production life, which reflects both the maturity of the system, as well as the improved user proficiency and experience. Second, we observe that the number of job submissions in 2016 far exceeded the rest of the years. This coincided with a period when Titan experienced an increased number of GPU failures (stating in late 2015, peaking in 2016, and ending in early 2017) [7], [8], resulting in an increased number of job failures. These failures were mostly due to a manufacturing defect (a non-ASR resistor being used in the GPU board assembly), where the result was the formation of silver-sulfide on these components causing an electrical open circuit. Once triggered, this was a permanent fail-stop failure, and the remedy was to replace the NVIDIA SXM/GPU. For these errors to manifest, GPUs were not required to be actively in use - simply having an electrical current present on the component was sufficient. Further details on the nature, effects, and remedy of these failures can be found in [7].

As a consequence of these GPU failures, the GPU-enabled jobs (and also CPU-only jobs up to a degree, if a failed GPU happened to be physically in a given job's allocation) were impacted. As a result of this, users altered their behavior. Some users chose to run smaller scale jobs in hopes of avoiding the "bad" GPUs (this also contributed to the increase in the number of jobs). Some users allocated 1-2 spare nodes and set the job launcher's automatic retry flag that uses a spare node and restarts the process. Some users may have tried both strategies within the same job.

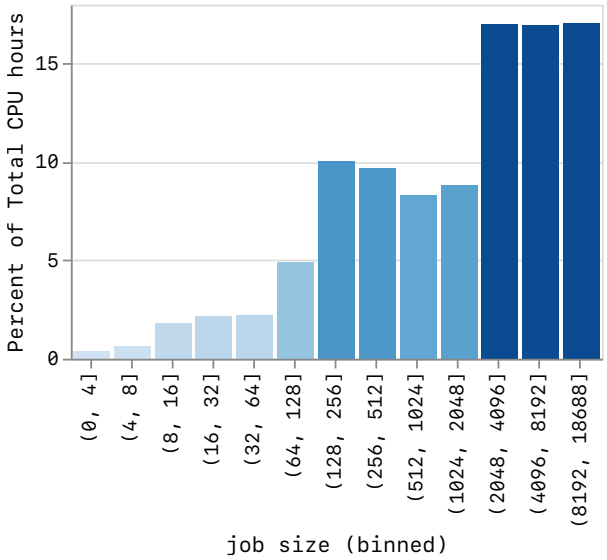

Fig. 1. CPU time allocation (percent) vs. binned job sizes across the five-year production period

III. RUR DATA ANALYSIS

Our analysis of RUR data focuses on three aspects of critical system resources: CPU time and memory, GPU time and memory, and I/O.

A. CPU and Memory Usage Analysis

Titan's CPU time is reported by *stime* (system time) and *utime* (user time) metrics. In addition, Job *start_time* and *end_time* metrics are also reported. Combined with the *node_count* data, we can then deduce the actual core-hours spent on a given job. Based on our analysis, 60% of the jobs are 1-node jobs, and roughly 90% of jobs fall below 256 nodes - this is understandable given most of the development and testing happens at the smaller scale. Figure 1 shows the distribution of total CPU hours percentage with respect to the binned job-sizes across the five-year production period. We clearly observe that the majority of the CPU hours were spent on medium- to large-scale jobs ($1/5^{th}$ to full Titan scale jobs). Further analysis shows that, across the five-year production period, Titan consistently spent more than 54% of the CPU hours on job sizes equal to or larger than 2,048 nodes.

To analyze the memory usage, we utilize the RUR parameter *max_rss* which reports an **estimate** of the maximum resident CPU memory used by an individual compute node through the lifespan of a job run: it may not reflect the true peak resident memory size if the memory consumption decreases from the peak allocated amount during the job runtime on a

TABLE II
MEMORY USAGE PROFILING

Year	Mean of rss_max (MiB)	Max of rss_max (MiB)	Percent
2015	680.96	29902.63	2.08%
2016	798.72	30624.80	2.44%
2017	586.68	30591.70	1.79%
2018	2022.00	30596.95	6.17%
2019	573.44	30480.81	1.75%

TABLE III
MEMORY INTENSIVE APPLICATIONS

Year	# of Total jobs	# of Unique apps	Top 5 apps
2015	2,606	115	51.66%
2016	7,299	127	63.28%
2017	8,380	150	61.85%
2018	6,415	127	52.31%
2019	1,741	88	50.54%

given compute node. For completeness sake, we mention here that each Titan compute node is equipped with a 16-core CPU with a total of 32 GiB CPU memory, and every CPU is paired with a single GPU with a 6 GiB GPU memory.

Figure 2 shows a profiling of the average memory usage binned by the node count or job size, across five years. We make two observations. First, the general trend is such that memory usage increases as the job size grows. This observation is supported by the Pearson product-moment correlation coefficients. These were calculated as 0.69, 0.74, 0.83, 0.80, 0.83, from 2015 to 2019, respectively. These positive correlations show that memory usage increases as the job size grows. Second, we observe that the average memory usage is around 4.5 GiB for large-scale jobs, which is only a fraction of the available CPU memory, but it is close to the 6 GiB GPU memory limit. We infer that many applications would allocate CPU memory less than or close to what the GPU memory limit is for facilitating data transfers to and from the CPU, which would explain the close affinity of these two values.

Table II provides a more quantitative summary on the mean and maximum CPU memory usage (derived from rss_max) for each year, as well as the usage percentage, i.e., rss_max out of the total available physical CPU memory (32 GiB). We note that there are certainly some applications using the maximum amount of CPU memory, which suggests that they are indeed memory bound. That said, the observed average memory usage is very low, which is consistent with observations made in [9].

Figure 3 provides a further analysis of the memory usage in quantiles (50%, 75%, 95%), and suggests that 95% of the jobs were using 4 GiB or less CPU memory (the 2018 data point is a clear outlier). Figure 4 provides a closer look at the daily average CPU memory usage in 2018. From late June to August of 2018, we see an influx of memory intensive applications that drove up the overall memory usage. The peaks in March and July correspond to the initial Gordon Bell submission deadline and the finalists' scaling runs, respectively.

Considering 50% of total physical memory (32 GiB) as the threshold, we analyzed the *memory-intensive* applications, and these results are summarized in Table III. The second column shows the total number of *memory intensive* jobs that represent an extremely small fraction of total number of jobs (i.e., 0.2% to 0.3%). Among these jobs, we identified the number of unique applications (provided in the third column) based on the application binary names and working directories data. The fourth column details the number of runs (as a percent) of the top five applications compared to the entire population of memory-intensive application runs. We observe that this set is extremely small, as the top five applications account for more than half of the total runs - many of these were repeated runs over 350 times per year. We consider this observation to be another characteristic of the leadership computing workload - there is a clear concentration of limited selection of leadership scale applications taking a large percent of platform runs.

B. GPU Time and GPU Memory Analysis

The *gpustats* plugins [1] provide GPU data in three values: *maxmem* represents the maximum memory used across all nodes; *summem* is the total memory used across all nodes, and *gpusecs* is the time spent on GPUs. However, for most sites, this data is not enough to capture the GPU usage. At OLCF, in particular, two extra modes are present via *gpu_mode* field that indicates how GPUs are used. Without any special flag, the GPU is in an *exclusive mode*, where only one process can operate a context to the GPU. With a special flag, an application can request the *default mode*, where multiple processes can communicate with a GPU. Accounting for the GPU usage is challenging for the default mode, because the GPU usage time can be inflated to n-fold, where n represents the number of ranks launched on a single node. Furthermore, the exclusive mode also presents an accounting challenge, where the NVIDIA GPU provides a functionality called the *multiple process service* (MPS) [10]. With MPS, an application can have multiple ranks leveraging a GPU through a proxy service. However, in reality only one process communicates to the GPU (the proxy), thus logically equating this to the exclusive mode. Our analysis shows that only 0.36% of all GPU-enabled jobs used special flags to enable the default mode, and therefore, we focus only on the exclusive mode.

Figure 5 gives an overview on GPU-enabled jobs over the five year period (2019 only has seven months of data). Looking at the percentage of jobs using the GPU each year, the peak usage (28%) occurs in 2018 as Titan itself enters the most mature and productive year. On average, we observe that 1/5th of total jobs are GPU-enabled. However, this does not paint a fair picture of the Leadership Computing mission at OLCF. If we zoom into the job sizes and re-analyze the GPU-enabled jobs, as in Table IV, we reach a very different conclusion.

The percentage of GPU-enabled jobs generally increases as the job sizes scale up. It reaches as high as 76% and 77% of large-scale jobs in 2017 and 2018, respectively. The decrease in the GPU/CPU usage ratio (see Fig 5) that occurred in 2016

[1] http://pubs.cray.com/content/S-2393/CLE%206.0.UP01/xctm-series-system-administration-guide-cle-60up01/the-gpustat-data-plugin

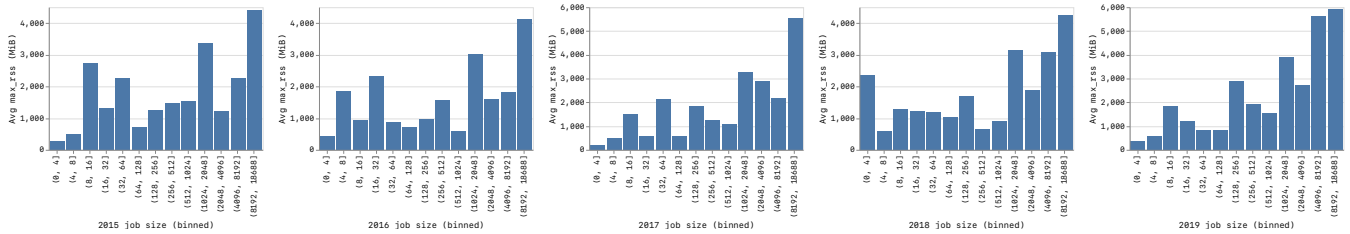

Fig. 2. Average memory usage over five years

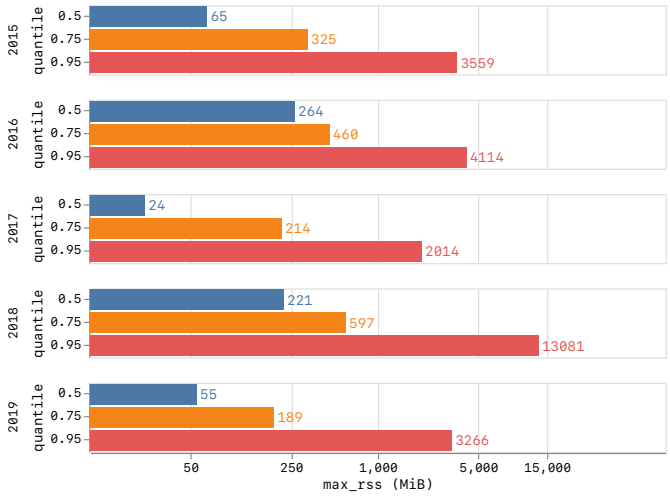

Fig. 3. Memory usage quantile across five years

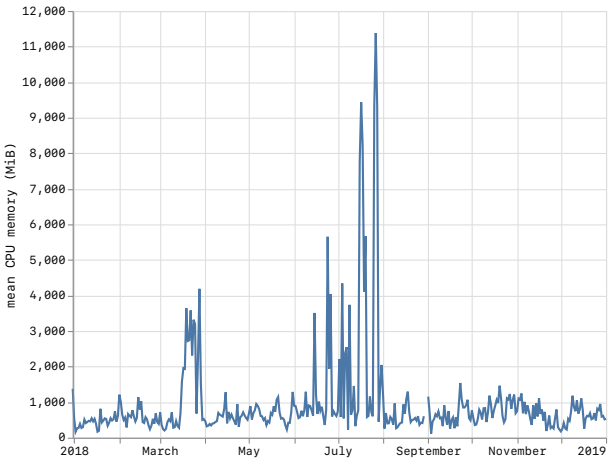

Fig. 4. Daily average CPU memory usage in 2018

was caused by GPU reliability issues that we have discussed earlier. During the late 2015 to early 2017 period, the larger sized jobs tended to fail more often due to GPU errors, and we observed through discussions with users and help tickets that users naturally refrained from submitting large-scale jobs. The 2019 data shows a drop in the largest jobs and we posit that these users were migrating over to Summit after it came online in January 2019.

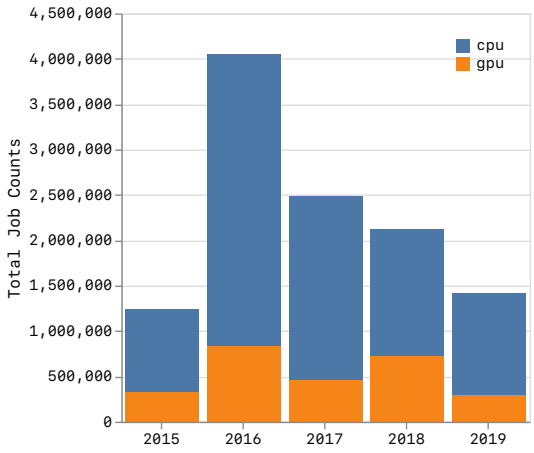

Fig. 5. GPU usage across five years

TABLE IV
GPU-ENABLED LARGE JOB ANALYSIS

Year	Job size, K=1024			
	[2K,4K)	[4K,8K)	[8K, 10,000)	[10,000, 18688]
2015	39.36%	56.86%	57.34%	58.50%
2016	53.24%	63.39%	40.90%	41.58%
2017	44.14%	54.78%	64.84%	77.87%
2018	45.27%	68.38%	67.52%	76.69%
2019	43.49%	65.38%	30.86%	48.26%

C. I/O Analysis

Traditionally, storage systems serving large-scale computing platforms are designed for sequential I/O workloads with large I/O sizes. One of the most dominant and critical I/O workloads on such platforms is defensive checkpoints which provide for a restart capability in case of node or job failures. With checkpoints, an application takes a snapshot of its state and persists it to the file system. In the event of a node or job failure, the application can read the checkpoint and restart the computation from a known prior state. As such, there is a long-held belief that such a simulation system will generally produce a write-heavy I/O workload. Even though it was anticipated that users/applications can conduct post data analysis on such systems (generating a read-heavy I/O workload), this was not expected to alter the system-wide overall I/O workload towards an aggregate read-heavy state.

There have been earlier studies analyzing the number of read/write requests from the storage system's (i.e., parallel file

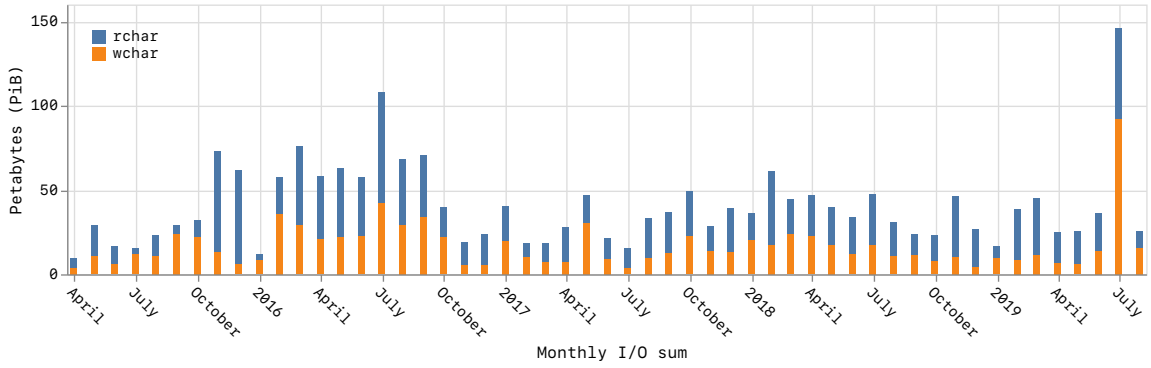

Fig. 6. Monthly data read or written over five years

system) perspective [5] and clearly pointing to an aggregated write-heavy I/O workload. The particular storage system that was used in this work was a shared-resource serving multiple computational platforms, including Jaguar (Titan's predecessor). At any given time, multiple applications were running on each platform, and the storage system observed an aggregated mix of I/O from all applications running on all connected platforms. Titan's RUR data, on the other hand, provides a different view from the perspective of applications that ran on Titan. Therefore, the storage-system and application views are providing different pieces of the same puzzle and both are needed to understand the complete I/O picture.

Figure 6 shows a time series breakdown of monthly I/O usage across the five-year period. Two fields, *rchar* and *wchar*, are monitored for each application run, respectively denoting the bytes read and bytes written **per process**. Noting that since these two metrics report bytes read and written during the entire job run and HPC I/O workload tends to be extremely bursty, it could be that the average bandwidth is low, but with bandwidth spikes across the session. Therefore, we cannot in general calculate any I/O bandwidth (or I/O rate) information based on these metrics.

Overall, we notice that the amount of reads and writes fluctuate, and it is not exactly obvious how we can infer any seasonal trend out of the pattern on per year periods. However, from Figure 6, we counted that in 44 months out of 61 months, read volume exceeds the write volume. It is even more apparent if we analyze the overall read to write ratios on a yearly basis, as given in Table V. On average, we see 44% more read I/O than write[2].

In Figure 6, we see that the overall amount of I/O is increased in 2016 compared to the remaining four years. The increase in I/O activity holds true for both reads and writes. This period overlaps with the increased GPU failures and perhaps a correlation between the two can be established (e.g., applications checkpointed frequently to guard against GPU failures and restarted more). This observation requires further analysis to provide a conclusive answer.

[2]The captured read data also includes I/O issued for gathering application binaries and shared libraries

TABLE V
YEARLY READ/WRITE RATIO

Year	Read (PiB)	Write (PiB)	Ratio (R/W)
2015	185	116	1.60
2016	395	287	1.37
2017	213	159	1.33
2018	278	163	1.70
2019	188	153	1.23

IV. CONCLUSION

Titan was a large-scale computational platform that was retired in August 2019 and has provided a wealth of resource utilization data. Our work provides a first-order analysis of this data spanning five years of operations.

We observed that the per compute client CPU memory usage increased with increasing job sizes (number of compute nodes). Also, we noticed that the average per client CPU memory usage was similar to the GPU memory capacity, perhaps suggesting that the CPU memory was used as a staging area for GPU memory. Our analysis also showed that GPU usage increased with respect to the increasing job sizes. Another interesting finding was the read-dominance of the I/O issued from Titan.

Our future work will focus on providing a similar analysis periodically from OLCF's new Summit supercomputer.

ACKNOWLEDGMENTS

We thank the anonymous reviewers whose comments have greatly improved this manuscript. We also thank Scott Atchley and Ross Miller for their comments and corrections. This research used resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at the Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725. This work was supported by the United States Department of Defense (DoD) and used resources of the Computational Research and Development Programs at Oak Ridge National Laboratory.

REFERENCES

[1] Andrew Barry. Resource utilization reporting - gathering and evaluating HPC system usage. In *Proceedings of Cray User Group Conference (CUG)*, 2013.

[2] Arthur Bland, Wayne Joubert, Don Maxwell, Norbert Podhorszki, Jim Rogers, Galen Shipman, and Arnold Tharrington. Titan: 20-petaflop Cray XK7 at Oak Ridge National Laboratory. In *Contemporary High Performance Computing*, pages 399–420. Chapman and Hall/CRC, 2017.

[3] Arthur S Bland, Jack C Wells, Otis E Messer, Oscar R Hernandez, and James H Rogers. Titan: Early experience with the Cray XK6 at Oak Ridge National Laboratory. In *Proceedings of cray user group conference (CUG 2012)*, pages 3–4. Cray User Group Stuttgart, Germany, 2012.

[4] Cray Inc. XC Series System Administration Guide: Resource Utilization Reporting. https://pubs.cray.com/content/S-2393/CLE%206.0.UP05/xctm-series-system-administration-guide/resource-utilization-reporting (visited August 2019).

[5] Y. Kim, R. Gunasekaran, G. M. Shipman, D. A. Dillow, Zhe Zhang, and B. W. Settlemyer. Workload characterization of a leadership class storage cluster. In *2010 5th Petascale Data Storage Workshop (PDSW '10)*, pages 1–5, Nov 2010.

[6] Jim Rogers. Measuring GPU usage on Cray XK7 using NVIDIA's NVML and Cray's rur. In *Proceedings of Cray User Group Conference (CUG)*, 2014.

[7] Jim Rogers. Statistical analysis of Titan's reliability as it reaches end of life. In *Cray User Group*, 2019.

[8] D. Tiwari, S. Gupta, G. Gallarno, J. Rogers, and D. Maxwell. Reliability lessons learned from GPU experience with the Titan supercomputer at Oak Ridge leadership computing facility. In *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, Nov 2015.

[9] Darko Zivanovic, Milan Pavlovic, Milan Radulovic, Hyunsung Shin, Jongpil Son, Sally A. Mckee, Paul M. Carpenter, Petar Radojković, and Eduard Ayguadé. Main memory in HPC: Do we need more or could we live with less? *ACM Trans. Archit. Code Optim.*, 14(1):3:1–3:26, March 2017.

[10] NVIDIA Developer Zone. Multi-Process Services. https://docs.nvidia.com/deploy/mps/index.html (visited July 2019).