Analyzing HPC Monitoring Data With a View Towards Efficient Resource Utilization

Samuel Maloney*, Estela Suarez*†, Norbert Eicker*‡, Filipe Guimarães*, Wolfgang Frings*
*Jülich Supercomputing Centre, Forschungszentrum Jülich, Germany
†Institute of Computer Science, University of Bonn, Germany
‡School of Mathematics and Natural Sciences, University of Wuppertal, Germany
Emails: {s.maloney},{e.suarez},{n.eicker},{f.guimaraes},{w.frings}@fz-juelich.de

***Abstract*—Compute nodes in modern HPC systems are growing in size and their hardware has become ever more diverse. Still, many HPC centers allocate the resources of full nodes exclusively to avoid contention, despite the associated risk of underutilization. This paper describes a thorough resource utilization study of CPU and GPU compute and memory capacity, and interconnect bandwidth on JUWELS, a mature leadership-class modular supercomputer, with the aim of identifying opportunities for improving utilization through advanced scheduling and node sharing. Separate analysis of CPU-only and GPU-accelerated nodes finds that CPU compute usage is already close to optimal for the CPU-only nodes, whereas there is plenty of scope for co-scheduling CPU-based jobs on GPU-accelerated nodes. Memory capacity and node-level interconnect bandwidth are sufficient to provision co-scheduled jobs. We analyze multiple one-month datasets to validate robustness of conclusions over time and compare with previous studies on other systems to establish generalizability of results.**

***Index Terms*—High performance computing (HPC), Resource management, Monitoring, Dynamic/Adaptive scheduling, Predictive analytics**

I. Introduction

In the pursuit of ever more computational power and—perhaps more so in the future—energy-efficiency, the compute nodes in modern high performance computing (HPC) systems are growing in size and their hardware has become ever more diverse, with nine of the top ten systems on the November 2023 TOP500 list now being GPU-accelerated [2]. However, this heterogeneity presents challenges for both application developers as well as system administrators in how to ensure efficient utilization of the various resources in the nodes.

Many HPC centers allocate the resources of full nodes exclusively, while codes are likely to bottleneck on only one (or a small subset) of the available devices, leaving the remaining underutilized. Co-scheduling of jobs (a.k.a. colocating or node sharing) is one potential option to improve overall utilization by concurrently running jobs with complementary resource needs on the same nodes. Nonetheless, it is often not used by default because of the risk for severe performance degradation due to contention when jobs with overlapping requirements end up co-scheduled and competing for the already bottlenecked shared resource. Mitigating this risk requires careful monitoring, making robust implementation much more difficult.

The primary contributions of this paper are

- a thorough study of resource utilization of CPU and GPU compute and memory capacity, and interconnect bandwidth on JUWELS, a mature leadership-class modular supercomputer;
- identification of opportunities for improving utilization through node sharing, as well as which resources are already well-used;
- analysis of multiple one-month datasets to validate robustness of conclusions over time;
- comparison with previous studies on other systems to establish generalizability of results;
- release of the anonymized operational data and scripts used for the analysis.

A. Related works

A number of prior studies have looked into resource utilization of HPC systems [3]–[9]. Browne et al. [3] introduced an open source HPC monitoring system, Gómez-Iglesias et al. [4] presented a tool for user-level monitoring of individual applications, Nikitenko et al. [5] explored on-the-fly analysis and making monitoring data available to users, Wang et al. [6] gave a high-level summary of five years of usage data from the Titan system, and Netti et al. [7] discussed an operational data analytics framework for use in HPC facilities.

Several others focus more narrowly on memory utilization [10]–[13]. Turner and McIntosh-Smith [10] determined memory requirements of applications on ARCHER, Zivanovic et al. [11] analyzed the memory footprints of several HPC benchmarks and production applications, and Panwar et al. [12] quantified memory underutilization and propose new microarchitecture techniques for leveraging the unused capacity.

We build particularly on the works of Peng et al. [13], who investigated memory usage of the Lassen and Quartz systems; Michelogiannakis et al. [8], who looked into the potential for disaggregation in the Cori system; and Lie et al. [9], who characterized the Perlmutter system [14].

This work has received funding from the European Commission's H2020, and EuroHPC Programmes, under Grant Agreement number 955606 (DEEP-SEA). The EuroHPC Joint Undertaking (JU) receives support from the European Union's Horizon 2020 research and innovation programme and Germany, France, Spain, Greece, Belgium, Sweden and Switzerland. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS Supercomputer JUWELS [1] at Jülich Supercomputing Centre (JSC).

DOI 10.1109/SBAC-PAD63648.2024.00023

TABLE I
JUWELS MODULE CONFIGURATIONS

Cluster (standard CPU nodes)	**Booster**
CPU-only	4× NVIDIA A100, 40 GiB HBM2
2× Intel Xeon Platinum 8168	2× AMD EPYC 7402
96 GiB DDR4	512 GiB DDR4 (host)
2271 nodes	936 nodes
279 nodes per cell	48 nodes per cell
InfiniBand EDR fat tree	InfiniBand HDR Dragonfly+
Both have 48 physical / 96 logical cores per node	

Disaggregation of resources, particularly memory, shows great promise for reducing underutilization [8], [15]–[22]; however, in general it requires specialized hardware or even photonics to mitigate additional latency [15], or changes in user code to effect an implementation in software [21]. This work therefore focuses on the potential improvements using existing hardware via (dynamic) co-scheduling of applications with complementary resource utilization profiles.

Co-scheduling has been studied in various contexts [23]–[31]. Blagodurov and Fedorova [23] used on-the-fly process migration for contention-aware scheduling, Galleguillos et al. [24] developed a data-driven approach for job duration prediction for dispatching methods, Netti et al. [25] proposed new heterogeneity-aware resource allocation algorithms, Xiong et al. [26] explored colocating with oversubscription to increase overall throughput, Goponenko et al. [28] integrated monitoring data with Slurm for I/O-aware scheduling, Zacarias et al. [27], [29] used machine learning to select applications for colocation which minimize performance degradation, Tzenetopoulos et al. [30] developed an interference-aware scheduler with Kubernetes, and Xue et al. [31] investigated the effect of split locks on virtual machine (VM) placement.

Particular attention has been paid to contention for memory bandwidth [32]–[36]. Zhuravlev et al. [32] looked into thread scheduling for reducing contention, Xu et al. [33] proposed minimizing memory bandwidth contention by maintaining an average utilization target, Breslow et al. [34] studied job-striping of HPC workloads, Dauwe et al. [35] examined energy use when using neural networks to predict contention, and Kuity and Peddoju [36] proposed a model for data locality and memory bandwidth contention-aware container placement.

The current work adds to the above body of research, studying the utilization of resources in the production HPC system JUWELS, for which such results have never been published before. We analyze its operational data for potential to improve resource utilization through advanced scheduling techniques. In doing so we combine some of the approaches used independently in previous publications: job monitoring, data analytics of operational data, and scheduling policies.

B. System configurations

All data have been collected from JUWELS [1], [37], the Tier-0 HPC supercomputer housed at Forschungszentrum Jülich (FZJ) in Germany. The configuration of JUWELS is based on the modular supercomputing architecture (MSA) [38]–[41] comprising two separate but closely interconnected modules referred to as the *Cluster* and *Booster* [42]–[44]. A brief overview of the salient hardware details of the two modules is given in table I and elucidated further below.

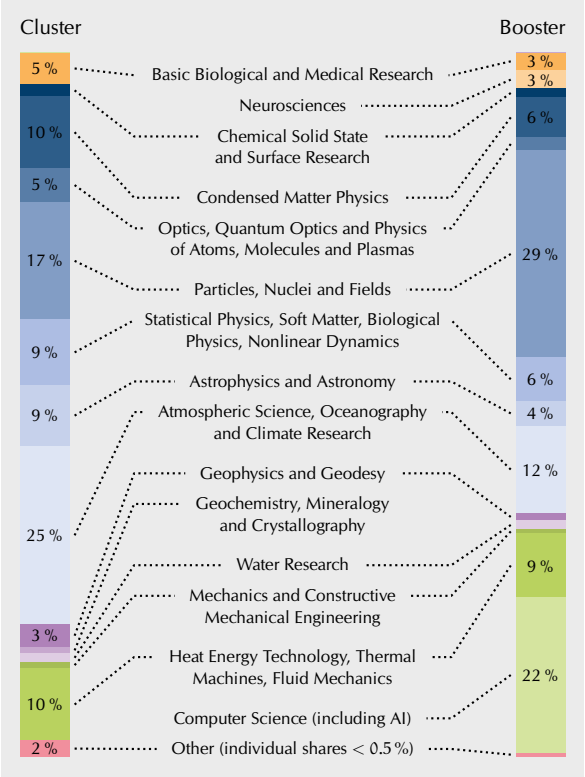

Fig. 1. JUWELS computing time allocations by research area, as of November 2023. Figure only includes categories with shares $\geq 0.5\,\%$ of total, and specific share amounts are only labeled for values $\geq 2\,\%$. Other research areas with shares $< 0.5\,\%$ on both modules include: Medicine; Agriculture, Forestry and Veterinary Medicine; Molecular Chemistry; Mathematics; Materials Engineering; Systems Engineering; and Electrical Engineering and Information Technology. For the Cluster, only allocations for standard (i.e., not GPU-accelerated) nodes are included, to better match with our datasets. The same allocations are in effect for both November and January datasets (see section II).

Users from many scientific domains run computations on JUWELS, so Fig. 1 gives an overview of the computing time shares allocated on JUWELS as of November 2023, broken down by research area. Approved allocations start on the first of May and November each year, so all of our data (discussed further in the next section) originate from the same allocation period illustrated in the figure.

The Booster module contains only GPU-accelerated compute nodes, of which there are 936, organized into 39 racks of 24 nodes, with 2 tightly coupled racks forming an InfiniBand cell within the Dragonfly+ topology, which is the basic building block of the network. Every node is equipped with four NVIDIA A100 GPUs, with each GPU having 40 GiB of HBM2 memory and a dedicated 200 Gbit/s InfiniBand NDR200 HCA. Every node also has two AMD EPYC Rome 7402 CPUs, which share 512 GiB of host DDR4.

The Cluster module has a more heterogeneous node configuration, with a primary set of 2271 CPU-only standard compute nodes supplemented by smaller sets of 240 large-memory nodes (also CPU-only), 56 GPU-accelerated nodes, and four visualization nodes. To keep the analysis more easily interpretable we restrict our dataset to just the standard compute nodes, which are each equipped with two Intel Xeon Platinum 8168 CPUs sharing 96 GiB of DDR4. Nodes on the Cluster are organized into cells comprised of three physical racks (two racks for compute and one rack for the associated network switches) with each cell having 279 nodes.

Both CPU types have 24 physical cores per socket, with Intel Hyper-Threading (HT) Technology or AMD Simultaneous Multithreading (SMT) enabled, as applicable. This gives a total count of either 48 physical or 96 logical cores per node on both modules. Resource allocation and scheduling is performed by the Slurm workload manager [45]–[47] (version 22.05.9 at time of writing) and node sharing is not currently used on either module, so jobs have exclusive access to all resources on the nodes in their allocation.

We note that throughout this paper we may refer interchangeably to jobs running on the Cluster module as "CPU" or "CPU-only" and jobs running on the Booster module as "GPU" or "GPU-accelerated", regardless of actual GPU utilization.

II. Data collection

Job monitoring information for JUWELS is collected and managed by a reporting and visualization tool called LLview [48], [49], developed in-house at FZJ. LLview is formed from a collection of software that oversees the acquisition of hardware status and usage data from the various components of each system at FZJ, and relies on the system monitoring stack deployed by the operations team. The monitoring stack primarily uses Prometheus [50] for collecting hardware metrics on compute nodes and IBMS (a fabric monitoring system tool by Eviden) for InfiniBand monitoring. LLview collates the different monitoring data and maps it to job submission and scheduling data from Slurm, and then provides a web interface for users and system administrators to view and explore the data in detailed interactive reports.

Most of the data are sampled by LLview at one minute intervals, on average, and are available in the live web portal for three weeks before being archived. This sampling interval does mean that very short transient features of the time series may be missed or averaged out, but it still provides plenty of data for many system-level statistical analyses, especially considering the small overall contribution made by very short jobs with few data points, as noted in section II-A.

Data for the GPUs (compute and HBM2 utilization and power draw) are collected using the NVIDIA Management Library (NVML) [51], which is queried every minute by a daemon developed in-house. Data for CPU compute and host DDR4 utilization are collected from Slurm as the average over the preceding minute.

There are two primary datasets presented in our analysis, consisting of all jobs that ended during either November 2023 or January 2024, respectively.[1] Jobs that were explicitly submitted to 'development' queues are excluded in order to focus on production workloads. The November dataset contains 39 858 and 47 332 jobs on the Cluster and Booster, respectively, while for the January dataset there are 73 335 and 46 837.

Maximum and average resource utilization for a job are defined by applying the respective aggregation function over both the time series for each node and over the nodes allocated to a given job; for the GPU-specific data, the aggregation is instead over both the time series for each individual GPU and over all the GPUs allocated to a given job.

Analysis of the data was carried out in Python using the pandas module [52], [53]. In order to gain useful insights from the large volume of data, our primary tool is the cumulative distribution function (CDF), which gives the cumulative percentage of jobs—either by count, node-hours, or GPU-hours; as indicated in individual figures—for which the quantity of interest is $\leq x$. CDFs using node-hours/GPU-hours are particularly useful for high-level resource utilization analysis because the plot area then represents the total resource-hours available in the given time period, and the areas above and below the CDF curve provide insight into the utilized and idle fraction of said resource-hours, respectively.

A. Job Durations

Fig. 2 shows a full breakdown of the distribution of jobs by runtime, with CDFs for both the number of jobs by count (dotted lines) and weighted by node-hours (solid lines).

On both modules a majority of jobs by count are short (< 1 h) but this group represents only a small fraction (~5 %) of system node-hours. Such short jobs are often assumed to be mainly test jobs (such as parameter scans), debugging jobs, or failed jobs, and therefore not indicative of real system usage. As such, some studies exclude short jobs from their analysis, but we have elected to retain them in this study for two main reasons: firstly, the choice of cut-off runtime is effectively arbitrary (e.g., Peng et al. [13], Lie et al. [9], and Michelogiannakis et al. [8] all use different thresholds of one minute, one hour, and two hours, respectively); and secondly, the small contribution of node-hours for short jobs means that simply weighting the analyses by node-hours organically reflects the true system impact of jobs of all durations in any case.

Both modules have a maximum wall clock time of 24 hours, and there is a noticeable grouping of 10-20 % of the node-hours near this upper bound for both modules. On the Cluster, the distribution of runtimes is otherwise quite even throughout the entire range, while on the Booster a distinct kink in the plotted CDFs of Fig. 2 can be seen at the six hour mark. This is likely a result of policy, where a shorter wall clock time limit of six hours is enforced for projects that have already exhausted their allocated quota for a given month. Users belonging to such projects may still submit jobs, but they are scheduled with

[1] Subsets of data from other time periods were also explored in the process of developing the current study, with similar statistical trends being observed, but only results from the two main datasets are presented, as they are the only ones with complete data for all resources studied.

Fig. 2. Cumulative distribution functions of job runtimes, both by count and weighted by node-hours. Annotated values are computed from the average of the November and January datasets. Majority of jobs by count are short (< 1 h) but this group represents only a small fraction of system node-hours. 24 h is the normal wall clock time limit on both modules, while 6 h is the wall clock time limit for projects that have already exhausted their allocated quota, with a visible kink at 6 h in the Booster data.

lower priority and are subject to the shorter time limit. That this kink appears only for the Booster would suggest that there may exist more projects on the Booster that are consuming more compute resources than were allocated to them.

B. Job Sizes

Fig. 3 gives a breakdown by the number of nodes requested per job. Approximately 50–60 % of jobs by count on both modules only request a single node, but even these small jobs contribute non-negligibly to the system node-hours, particularly on the Booster, with 6 % and 20 % of Cluster and Booster node-hours respectively coming from single node jobs.

Compared to Li et al. [9] we see fewer single node jobs, with 49 % and 57 % of CPU and GPU jobs, respectively, compared with their 68 % and 66 %. Some of this difference may be workload differences and statistical noise (particularly for GPU jobs the difference is not so significant), but there could also be an effect from our dataset being collected later in the life cycle of JUWELS. The Cluster was deployed in the summer of 2018 and the Booster in November 2020, giving 3–5.5 years between deployment and collection of our datasets, vs. only 1–1.5 years between Perlmutter's phased deployment in 2021/2022 [14] and the dataset used by Li et al. [9], collected November 1 to December 1, 2022. This may have given more time for users to scale up from single node development and testing.

Additionally, >99 % of CPU jobs by count (>74 % by node-hours) would fit within the 279 nodes of a single cell on the Cluster, while on the Booster, >98 % of jobs by count (>75 % by node-hours) would fit within the 48 nodes of a single cell. This is similar to the results of Li et al. [9] where they observed that >99.6 % of both CPU and GPU jobs would fit within the 256 or 128 nodes of a single rack of the respective Perlmutter partitions. However, while they did not indicate the node-hour contribution of the remaining large jobs, we note that for our datasets these multi-cell jobs still contribute up to 25 % of the node-hours. This is useful knowledge for scheduling the nodes of a job close to each other within the network topology, as well as for potential future resource disaggregation strategies, where pooling at sub-system rack/cell levels has been mooted [8].

Conversely, very few jobs request truly large node counts occupying a significant fraction of the full modules. We note that there are also possible scheduling policy effects at play here, namely, if users wish to run particularly large or full-module scale jobs (>1024 nodes on the Cluster or >384 nodes on the Booster) they must submit to special queues that only run in designated time slots coordinated with the user support team. Therefore, the presence or absence of such large jobs in the data could vary greatly depending on such arrangements being made during a given period of time. Although rare, such large jobs can obviously still contribute significant node hours, as indicated by the visible steps in the node-hour CDFs at high node counts of Fig. 3, despite the almost imperceptible change in the accompanying count CDFs.

III. Resource Utilization

A. Main Memory on the CPU

In the spirit of Li et al. [9] and Peng et al. [13] we briefly characterize the jobs by their maximum CPU DDR4 utilization, as shown in Fig. 4. Those studies labeled their categories as low (≤25 %), medium (25–50 %), and high (>50 %) intensity. We have made one further division, such that we rather have very low (≤12.5 %), and low (12.5–25 %). We did this both to provide a bit more of a breakdown, but also because the Booster nodes have twice as much DDR4 as Perlmutter's GPU nodes, which means that in terms of actual memory values for GPU jobs, our very low category now corresponds directly to Li et al. [9]'s low category, our low with their medium, and our medium with their high.[2] All thresholds are determined from

[2]On the other hand, Perlmutter's CPU nodes have more than four times as much DDR4 as the Cluster nodes, so *all* our categories would fall below their low threshold.

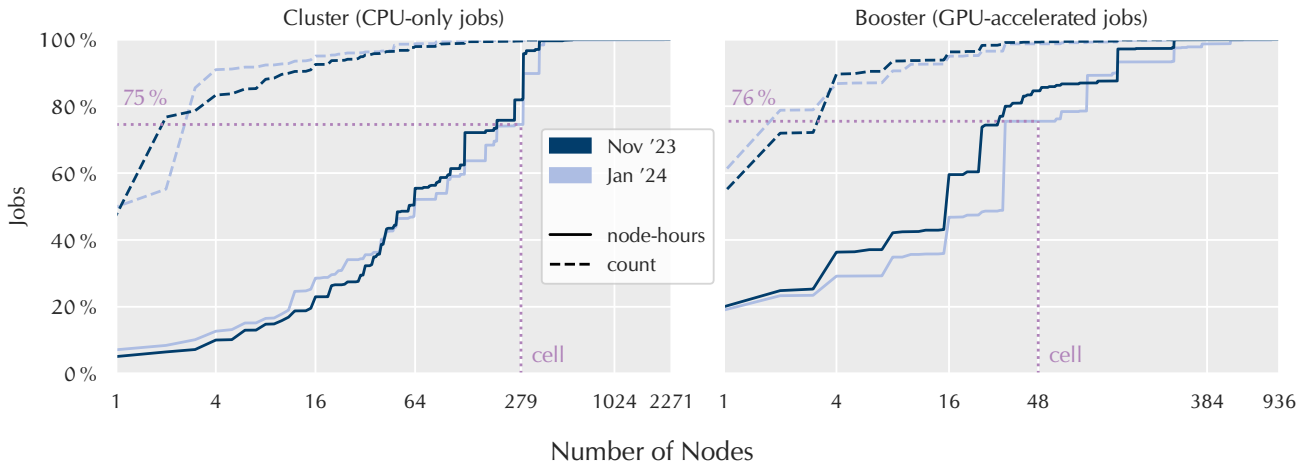

Fig. 3. Cumulative distribution functions of jobs sizes (i.e., number of allocated nodes per job), both by count and weighted by node-hours. N.B. log-scale on the x-axis. For the Cluster, 279 is the number of nodes per cell, while the Booster has 48 nodes per cell. Annotated values are indicated for the minimum of the November and January datasets.

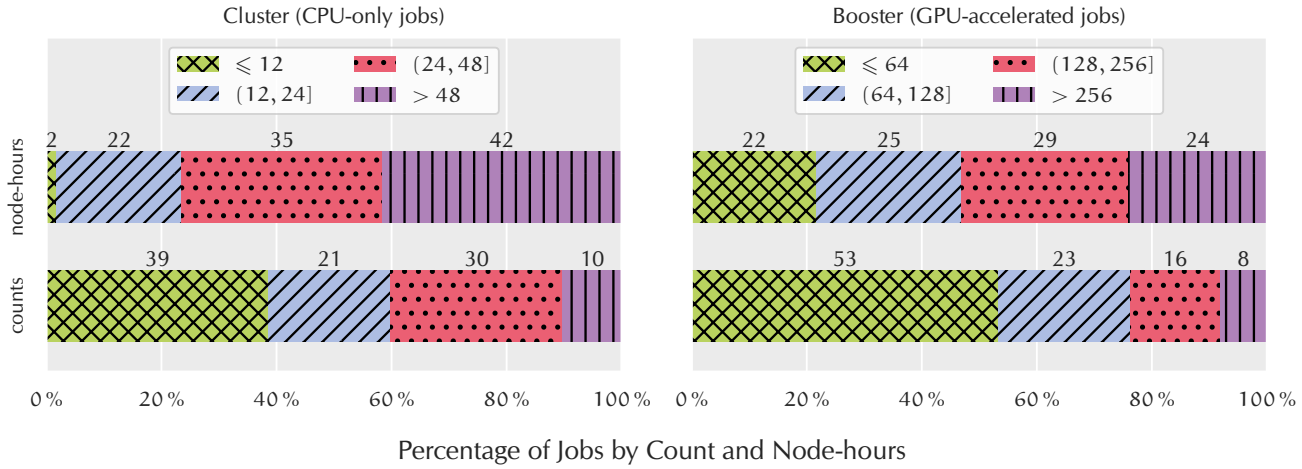

Fig. 4. Percentage of jobs by count and node-hours with maximum CPU memory utilization in the given ranges (in GiB). Jobs with high maximum CPU memory utilization tend to consume more node-hours. Data from both November and January datasets were combined to create this figure, although we note that the general trends are similar when they are plotted separately (not shown). Labeled percentages may not sum to 100 due to rounding.

the nominal DDR4 capacities for simplicity, and the specific values are given in the figure legend.

A similar result is seen as in Li et al. [9] and Peng et al. [13], where jobs with medium and high maximum memory intensity on both modules account for a greater fraction of node-hours than jobs by count. This indicates that jobs that reach a greater peak memory footprint are also more likely to run for a longer time and/or occupy a greater number of nodes.

For more granularity, Fig. 5 shows full CDFs of the CPU DDR4 usage per job, weighted by node-hours. The OS and other services running on the compute nodes invariably occupy some memory, with GPFS (General Parallel File System) in particular consuming several GiB, but we are only interested in the utilization rate for the remaining memory, i.e., that which is actually available to the application. Therefore, 100 % utilization is normalized to be the maximum value in the datasets for each module, as a more representative upper limit than simply the nominal hardware value. This assumption was manually validated by checking the logs for such jobs and noting that they showed "out-of-memory" errors, indicating that they had truly reached 100 % of the available DDR4. For reference, the actual maximum values observed were 90.84 GiB and 492.69 GiB on the Cluster and Booster, respectively, compared with the nominal values of 96 GiB and 512 GiB.

A majority of node-hours on both modules (~55 % on the Cluster and ~75 % on the Booster, averaged over both datasets) are accounted for by jobs that never consume more than 50 % of the DDR4 available to them. The CDFs for both modules have an initial flat section at the very lowest utilization levels, indicative of the background/OS memory usage setting a lower limit on the memory utilization seen in the dataset. For both datasets on the Cluster and the January dataset for the Booster, the CDF increases relatively consistently across the remaining range, while for the November dataset on the Booster a distinct plateau is seen, with the CDF becoming nearly flat between 60–80 % utilization. This might suggest that while memory is

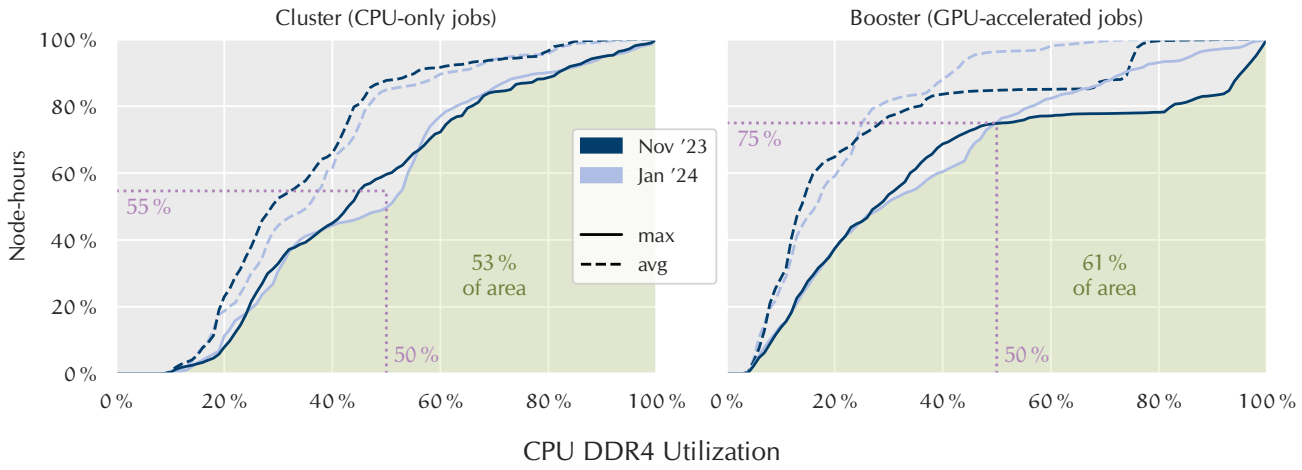

Fig. 5. Cumulative distribution functions of maximum and average CPU memory utilization per job, weighted by node-hours. 100 % utilization corresponds to 90.84 GiB on the Cluster and 492.69 GiB on the Booster. Annotated values in purple are computed from the average of the November and January datasets and the green area is bounded by the lowest CDF. Majority use less than 50 % capacity at maximum, and over half of all memory-hours are idle.

over-provisioned for almost three quarters of the node-hours in November, the remaining quarter form a distinct class of jobs that use most, if not all, of the the DDR4 available to them.

As mentioned in section II, the area of the plots represents the total system resource-hours available in the given time period, and so from a scheduling perspective, the area under the maximum CDF curves represents the fraction of the idle resource-hours that could be reclaimed given a perfect static allocation. What we mean by this is an allocation that is fixed over the runtime of a job, but is exactly large enough to meet the peak resource requirement of said job. Conversely, the area under the average CDF curves would represent the fraction of the idle resource-hours that could be reclaimed with a perfect dynamic allocation that changes to exactly match the resource requirements of the job at all points in its runtime.

Given that, one can then view the area between the maximum and average CDFs as an upper bound on the additional improvements achievable by considering dynamic vs. static allocations. Since the area below the maximum CDFs is significantly larger than the area between the maximum and average curves, it is clear that the bulk of theoretically possible improvements come from optimizing static allocations to better reflect peak demands. This still requires predicting what the maximum utilization for each job will be, an already non-trivial problem, but implies that the much harder problem of predicting the temporal evolution of each job's resource utilization to better match a dynamic allocation would only allow for marginal possible improvements over the static problem.

For both modules over 50 % of the plot area is below the curves, indicating substantial unused memory capacity for co-scheduling of jobs. However, we note that memory bandwidth is also an important potential source of contention, which we unfortunately do not have data for in this study.[3] Thus, one should implement a means to measure bandwidth usage and/or monitor carefully for performance degradation of co-scheduled jobs to determine those with individually high bandwidth usage. One possibility on modern architectures would be isolating jobs on separate NUMA (non-uniform memory access) domains and restricting their access to only domain-local memory, which should mitigate or even eliminate the risk of contention depending on the specific split of resources.

We note that certain comparisons between the datasets, such as the Kolmogorov–Smirnov test, are not particularly useful in this study because they are too sensitive to specific differences like the separations near 50 % in the Cluster CDFs or near 80 % in the Booster CDFs. In our case, the main conclusions stem directly from the plot areas and do not require the underlying distributions to be identical. The areas both under the maximum CDFs and between the average and maximum CDFs differ by at most 3 % between the datasets on both modules, with similar agreement for CPU utilization in the next section.

B. CPU Compute

Fig. 6 gives the CDFs of the CPU compute utilization weighted by node-hours. With HT/SMT enabled on both modules, 50 % on the x-axis implies only physical cores being used, while 100 % would indicate all physical and logical cores used at full capacity. On both modules we observe very few node-hours actually using HT/SMT, with only ~12 % of CPU and ~3 % of GPU node-hours reaching significantly above 50 % utilization.[4]

The large ~68 % step in the node-hours near 50 % utilization on the Cluster indicates over two-thirds of the jobs sitting right at this one-thread-per-core operating point. There is also a smaller step of ~9 % of the node-hours near 25 % utilization,

[3] Hardware counters for directly measuring memory bandwidth utilization are not available, and a proxy such as last-level cache misses is not currently collected, although we are investigating possibilities for future studies/systems.

[4] We manually choose a threshold just above 50 % by inspection as shown in Fig. 6 because background/OS processes result in usage slightly above 50 % even for jobs that target only the full complement of physical cores. For automated thresholding, 60 % would seem a reasonable compromise between capturing the top of the step but still avoiding much additional increase.

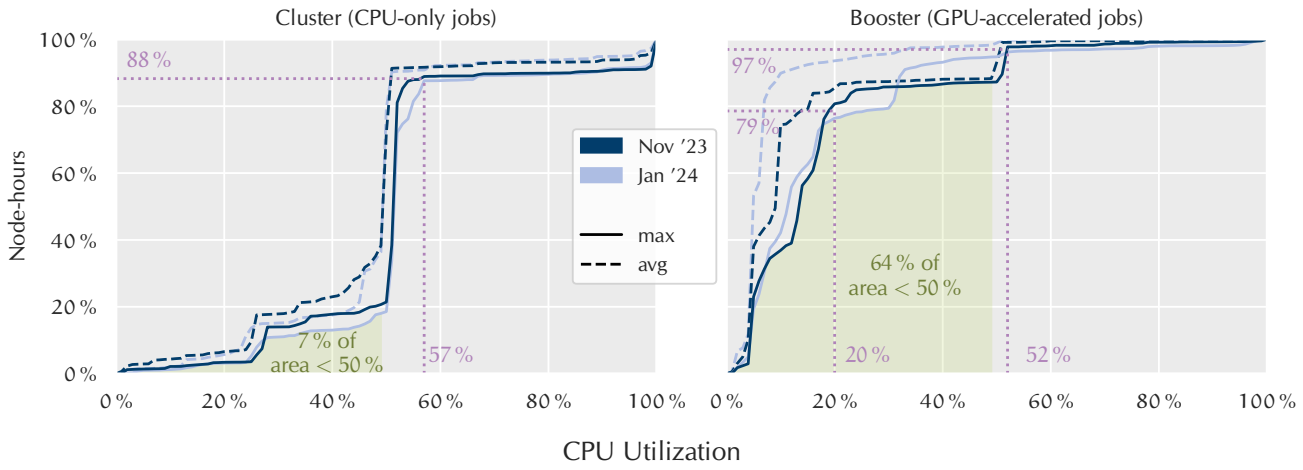

Fig. 6. Cumulative distribution functions of maximum and average CPU compute utilization per job, weighted by node-hours. 50 % implies all physical but no logical cores being used, while 100 % implies all physical and logical cores at full capacity. Annotated values in purple are computed from the average of the November and January datasets with the exact threshold for the top of the steps near 50 % chosen by visual inspection, see footnote 4. The green area is bounded above by the lowest CDF. Clear spikes around 50 % for both, and generally low utilization on Booster, where the CPUs primarily support the GPUs.

which could indicate particularly memory bound jobs for which only using half of the physically available cores provides an even better balance of compute to memory bandwidth, or where a smaller core count improves cache reuse.

From a scheduling perspective, the Cluster shows little room for potential improvement. While technically there is a large total area under the curves in the left plot of Fig. 6, in reality one would assume that codes running only on the physical cores are doing so because they are already bound by memory bandwidth, and so co-scheduling other jobs would almost certainly lead to contention (with the jobs near 25 % likely even more sensitive to contention from extra threads).

Given that a "perfect" utilization scenario would present as a CDF running flat along the bottom of the plot and then turning 90°upwards at the desired utilization, the sharp corners in the Cluster CDFs occurring at sensible operating points are about as close to optimal as one could realistically hope for. The average and maximum CDFs are also very close together, indicating little temporal variation in CPU utilization to exploit over the bulk of the node-hours.

In contrast, on the Booster almost 80 % of the node-hours reach maximally 20 % CPU utilization, demonstrating the merely supporting role played by the CPU on the GPU-accelerated nodes. LLview also computes a metric for the number of cores that are "in use" on a node, reported as a binary state for each individual core that is defined to be true iff its usage was above 25 % over the preceding minute. For the Booster, the median number of cores "in use" across both datasets is 4.2 per node (weighted by node-hours) indicative of the common use case where a single CPU thread is launched per GPU to coordinate computations.

Also of note is a small but clear step of ~10 % of the node-hours in both the average and maximum CDFs of the November dataset for the Booster near 50 % utilization. This indicates a class of jobs making full usage of the physical cores on this module, not just using the CPU as support for the GPU but actually performing sustained computations on the host.

With most of the plot area for the Booster under the CDFs (even considering only the left half of the plot below the 50 % utilization threshold) there is considerable room for co-scheduling of CPU consuming jobs on the Booster, especially in combination with the available DDR4 capacity seen in the previous section. Lack of memory bandwidth data would again be the major potential caveat, but for jobs that are well-adapted to offload the bulk of their workload to the GPUs, co-scheduling with CPU-only jobs should be a very tenable option.

Comparing with Li et al. [9], we observe similar overall conclusions, but a couple of differences stand out. They observed more CPU node-hours using SMT than we did (>20 % with >95 % utilization compared to our ~9 %). They also observed significantly more CPU intensive jobs on their GPU-accelerated nodes than we did, with >30 % reaching >55 % utilization on Perlmutter compared to our ~3 %. This could perhaps be due to the JUWELS system being in operation for a longer time, with users better off-loading the computational work to the GPU accelerators.

C. GPU Utilization

GPU compute utilization is reported by NVML as the "percent of time over the past sample period during which one or more kernels was executing on the GPU." [51] This does limit the utility of this measure for determining how efficiently a given GPU is being used, but does still allow one to glean whether at least some computation is being executed on a given GPU. For this reason (and to facilitate comparison with the results from Li et al. [9]) we report in Fig. 7 the utilization of the compute and HBM2 memory resources per GPU for each job, rather than per node, and weighted by the runtime to give GPU-hours rather than node-hours.

Similar to Li et al. [9], we see a small portion (~8 %) of the GPU-hours recording no kernels whatsoever executing during the job runtime. This is slightly less than the ~15 % of fully

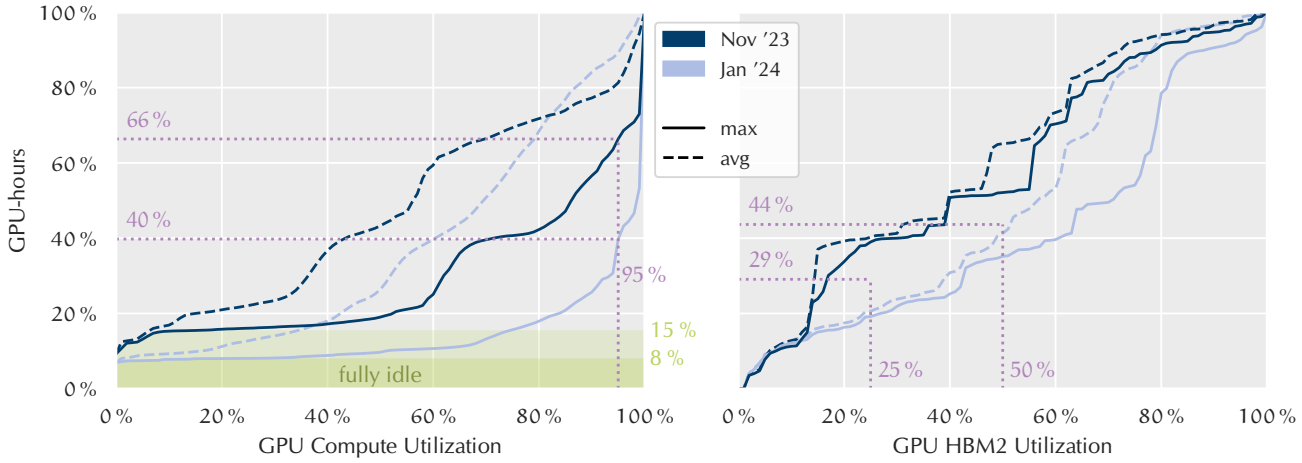

Fig. 7. Cumulative distribution functions of maximum and average GPU compute and memory utilization per GPU per job, weighted by GPU-hours. Annotated values in purple in left plot indicate the individual CDF values and in the right plot are computed from the average of the November and January datasets. Compute utilization is quite high, but it should be noted that utilization is measured as the "percent of time over the past sample period during which one or more kernels was executing on the GPU." [51]

idle GPU-hours reported by Li et al. [9], although if we include the entire grouping of GPUs with $<10\,\%$ utilization seen in the November dataset of Fig. 7, then our value becomes also ~15 % for November. The higher percentage of fully idle GPU-hours seen in the November dataset compared to January could be related to the observation in the previous section of a set of jobs heavily using the CPU in the November dataset that was not observed in January. We also hypothesize that some fully idle GPUs likely result from allocating full nodes, even though a given job may not be configured to use all 4 GPUs.

In general, the GPU resource curves exhibit a high degree of variation between the November and January datasets compared to the other resources analyzed. For November, we obtain a very similar CDF for GPU compute utilization as compared to Li et al. [9], with 34 % of our GPU-hours belonging to jobs reaching at least 95 % utilization compared to their 38.5 %. The January dataset exhibits significantly higher utilization with fully 60 % of GPU-hours at or above 95 %.

Identifying unused GPU memory resources is slightly more complicated, as none of the HBM2 utilization values in our dataset are truly zero (minimum 0.438 GiB), consistent with the NVML documentation indicating that "the driver/GPU always sets aside a small amount of memory for bookkeeping." [51] The maximum value was the nominal 40 GiB, so that is used as 100 % for normalization. After this normalization, the data were manually inspected and a grouping of data was observed with maximum utilization within 0.5 % of the minimum, so this is used as a threshold to denote jobs where the actual user code likely did not make any use of the GPU memory. By this measure, we observe ~3.8 % of GPU-hours with no HBM2 utilization—noticeably lower than for fully idle compute.

This is again lower than observed by Li et al. [9], who recorded 10.6 % of GPU-hours using no HBM2 capacity, but also matches their finding of more GPU-hours with idle compute as compared to GPU-hours with idle memory. The reason for this is not clear, but they posit that it may indicate memory being used by other GPUs via the NVlink or for some other purpose by the job. For non-idle memory, we also see relatively even distribution across the full range, albeit with higher overall utilization than Li et al. [9], with averages over both datasets of only ~29 % and ~44 % of GPU-hours using less than 25 % and 50 % of the HBM2 capacity, respectively, compared to the 49.9 % and 70.5 % observed by Li et al. [9].

While co-scheduling jobs on a single GPU would require great care to avoid contention issues, co-scheduling jobs onto different GPUs within a node should be much more straightforward, especially given the availability of CPU compute and DDR4 resources already observed. This would hopefully address the 8–15 % of fully idle GPU compute hours, and one would then not have to worry about HBM2 capacity or bandwidth contention because each job would have its own separate GPU hardware and dedicated HCA.

1) GPU Power Draw: We know that measuring energy usage of jobs will ultimately be crucial for characterizing the (energy) efficiency of HPC systems, but such energy data is still in short supply. In our current datasets, the only relevant metric is the power draw of the GPUs reported by NVML, which could in theory be integrated to estimate GPU energy usage for a job, but the one minute sampling interval of our data would make this a rough estimate. Fortunately, the GPUs account for the majority of the power draw on most accelerated systems, so even in the absence of data for other components it can still provide some useful insights. For reference, each A100 has a thermal design power (TDP) of 400 W while for each EPYC 7402 CPU the TDP is 180 W, meaning the four GPUs can draw over 4× as much power as the two CPUs, and as seen in section III-B the CPUs have low utilization on the Booster making it likely the real ratio is even higher.

CDFs of the measured GPU power usage are shown in Fig. 8. Of immediate note is that while the average CDFs show that virtually all jobs have an average draw within the nominal TDP, just over 10 % of the GPU-hours have maximums that

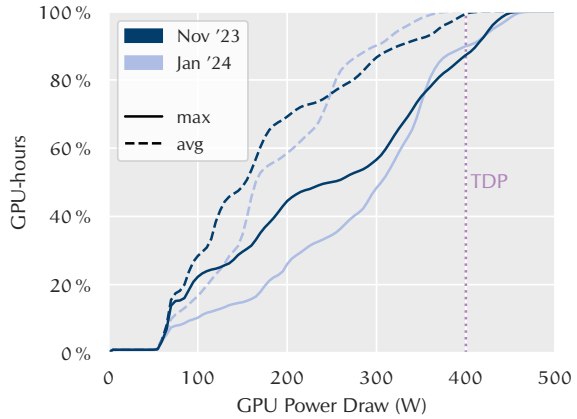

Fig. 8. Cumulative distribution functions of maximum and average GPU power draw per GPU per job, weighted by GPU-hours. Even fully idle GPUs consume power, and the shape of the curves are much more similar to the HBM2 Utilization in Fig. 7 than that of the utilization reported by NVML. Average power draw is almost fully within TDP but maximums may exceed it.

spike above this, with the maximum reported value being 479 W. Partly for this reason, and partly because power is not a resource we are trying to maximize usage of, we have not normalized the x-axis in Fig. 8, but rather left it in Watts.

Also, $< 1\,\%$ of GPU-hours had power draw close to zero, despite observing 8–15 % of GPU-hours with next to zero compute utilization in Fig. 7. This confirms that such hardware still consumes some power even while not in use, and therefore maximizing utilization and reducing idle time should be more energy efficient for a given amount of computational work.

Given the overly optimistic nature of the utilization metric reported by NVML, where even a single kernel running on the GPU shows it as in-use, the power draw can also provide an interesting comparison as a possible proxy for more realistically indicating how fully occupied the GPU compute units are. The curves in Fig. 8 are indeed much more similar to those of the HBM2 utilization in Fig. 7 and do not show anything like the saturation near 100 % of the compute utilization in the latter (Fig. 7). While far from perfect—especially since maximizing power usage is certainly not a target outcome in itself—this does indicate that, in the absence of better compute utilization data, the power draw might serve as a rough approximation.[5]

D. Interconnect Utilization

To quantify the bandwidth utilization of the network interconnect, LLview reports the total volume of data moved into and out of each node. These values will include both I/O data movement to and from the storage system, as well as communication with other nodes in the job. Data being communicated with other process on the same node would not be included here. On the Cluster the nominally reported network injection bandwidth is 12.5 GB/s bi-directional from a single node while for the Booster it is 100 GB/s [1], and so 100 % utilization is normalized to these values in Fig. 9.

Values reported by LLview are average rates of data moved over the preceding one minute sample period, so short bursts will be reduced by averaging. Therefore, the values of the maximum CDFs in Fig. 9 are the maximum values for each job recorded in our dataset, but instantaneous rates could have been significantly higher. In particular, note that sustaining 100 % of the bandwidth caps over a full one minute sampling interval would imply moving 750 GB or 6000 GB of data per node per minute for the Cluster and Booster, respectively. It would seem likely to be difficult for a code to make effective use of such volumes of data, particularly when the local memory capacity is almost an order of magnitude smaller. As network bandwidth utilization is known to be bursty [8], it is likely that many jobs will still max out the bandwidth over shorter intervals, so while it may seem over-provisioned on average—due to our large sample interval—that is because the limit is designed to avoid bottlenecking during intense bursts and thereby degrading application performance.

Fig. 9 indeed shows that the vast majority of the workload on both modules never require such high sustained data transfer. With the exception of incoming data transfer on the Cluster, over 80 % of the node-hours reach maximally 15 % of the respective maximum bandwidth per minute. That the maximum incoming data transfer is significantly higher on the Cluster compared to the outgoing, while the average CDFs for incoming and outgoing almost entirely overlap, implies that Cluster workloads tend to have short but intense phases of reading in data, presumably during initialization, whereas writing reaches significantly lower sustained peaks. Interestingly, this separation is not seen for the Booster, which implies a more balanced ratio of data being read and written throughout the runtime of GPU-accelerated jobs.

Michelogiannakis et al. [8] observed similarly low network bandwidth usage per node, although comparing exact numbers is problematic because firstly, they sample every second, which will facilitate capturing of short but intense peaks; and secondly, they plot their CDFs using all samples individually, rather than first aggregating over each job. They do see higher usage on their higher concurrency KNL (Knights Landing) nodes compared to their Haswell nodes, a trend that is borne out also in our data, where the Booster nodes reach much higher absolute bandwidth utilization compared to the Cluster (the entire range for the Cluster data is just the smallest 12.5 % of that for the Booster). This is to be expected, as the much greater computing power of the GPU accelerators would require commensurately more data to keep the compute units occupied.

For scheduling, this implies that one need not be too concerned about saturating the individual node interconnect bandwidth limits when adding more jobs; however, further investigation of other potential choke points in the network topology will be required beyond the scope of this present work. In particular, the Cluster CDFs for incoming data transfer have fat tails, which might merit some special consideration for high-utilization reading-in phases of certain jobs.

[5]We note that better measures of GPU occupancy might be gathered from tools such as NVIDIA's Data Center GPU Manager (DCGM) [54], but a proxy can still be useful when such tools are unable to be used (e.g., when the use of other profiling tools precludes it). FZJ is in the process of adding improved GPU measures to LLview, but this had not occurred in time for this study.

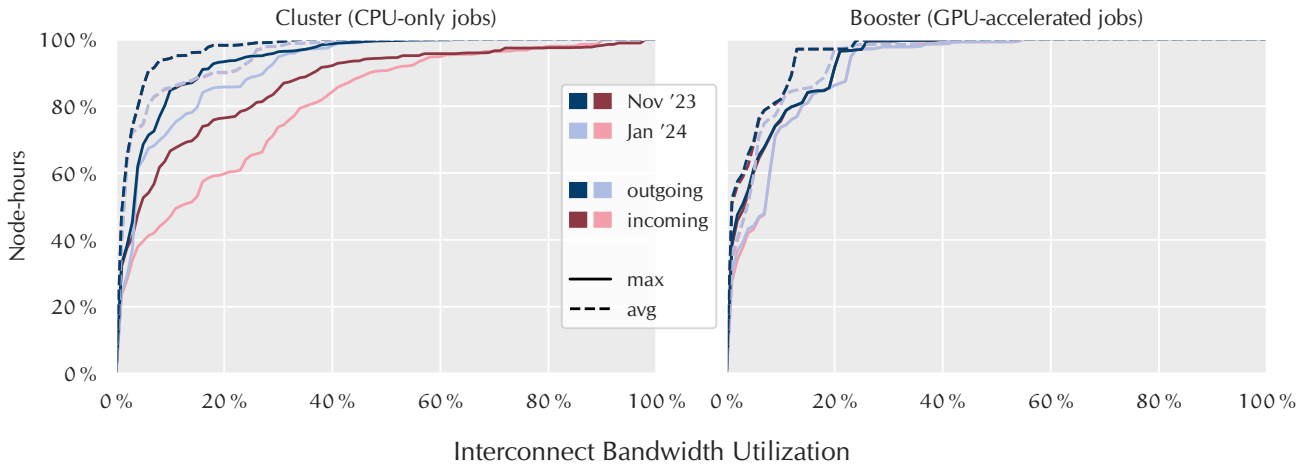

Fig. 9. Cumulative distribution functions of maximum and average outgoing and incoming interconnect bandwidth utilization per job, weighted by node-hours. Blue and red denote outgoing and incoming data transfer, respectively (in some cases the blue almost fully overlaps the red). Darker shades and lighter tints denote the November and January datasets, respectively. Utilization is measured as the average over the preceding minute, and 100 % utilization corresponds to 12.5 GB/s on the Cluster and 100 GB/s on the Booster. The data indicates very low utilization for all curves, with fatter tail for incoming data on the Cluster.

IV. Conclusions

This paper describes a thorough study of the utilization of various hardware resources on JUWELS, a modular supercomputer that has been in production for several years. We analyze two independent full-month datasets from JUWELS to verify the robustness over time and also compare with results from previous literature to establish that our main conclusions are generalizable between different machines and HPC centers.

Overall utilization is found to be much closer to optimal on the Cluster module, particularly for CPU compute, where almost all node hours occur at sensible operating points and the average and maximum usage are close to each other. This aligns with the hypothesis that high utilization is easier to achieve on more homogeneous systems, especially as application developers have had more time to adapt codes for multicore CPU-only architectures as compared to accelerated systems.

On the Booster, the CPUs are found to merely support the GPUs, as expected, so there would be plenty of scope for co-scheduling CPU-focused and GPU-focused jobs. There is also significant underutilization of host memory capacity, with only a quarter of jobs using more than half the DDR4 at any point in their runtimes. Thus, there would be capacity available for co-scheduled jobs (memory bandwidth must also be considered, as noted in section III-A, but was outside the scope of data available for this study). Some fully idle GPU-hours are observed, likely due to jobs that are not configured to use all four GPUs on a node, so co-scheduling of multiple such jobs on a single node would be an obvious first step to improve GPU utilization without having to worry about contention for individual GPU compute, HBM2 or HCA resources.

Analyzing the areas under and between the CDFs for both the average and maximum resource utilization revealed that most co-scheduling gains should be realizable by optimizing static allocations, which do not change over time but are matched to the maximum amount needed at any point in the job's runtime. Attempting to co-schedule jobs with complementary temporal variation (where the maximum requirements may be larger than can be accommodated concurrently, but are expected to occur at different times) is both a much more challenging problem and with much smaller potential gains.

We were also able to make many similar observations as Peng et al. [13], Michelogiannakis et al. [8], and Li et al. [9] despite having a one minute sampling frequency for our data, compared to their one and ten second cadences. Moreover, this validates the procedure of Peng et al. [13] where they coarsened their data to one minute intervals before analysis. This demonstrates that useful conclusions can be drawn without requiring intractably large data sets and while minimizing the impact of monitoring processes on the executing workloads. Higher sampling rates can be reserved for only selected resources such as interconnect usage, where bursty rather than sustained usage is more common.

Future work will include making such analysis more systematic, e.g., generating such CDFs at periodic intervals to better understand and identify high-level changes or anomalies in system utilization. As well, actual development, implementation, and testing of dynamic co-scheduling policies will be undertaken as part of a multi-year effort to improve efficiency, guided initially by the conclusions from this study. This will also involve addressing identified deficiencies such as how to measure memory bandwidth usage to avoid contention therein.

Acknowledgments

The authors gratefully acknowledge our colleagues at the Jülich Supercomputing Centre, in particular Benedikt von St. Vieth for comments on the draft manuscript and details of the hardware and monitoring infrastructure, and Andreas Herten for insights on the network configuration. We would also like to acknowledge George Michelogiannakis from Lawrence Berkeley National Laboratory and Jie Li from Texas Tech University for discussions during the early stages of this work.

References

[1] Jülich Supercomputing Centre, "JUWELS Cluster and Booster: Exascale pathfinder with modular supercomputing architecture at Juelich Supercomputing Centre," *JLSRF*, vol. 7, A183, Oct. 29, 2021, https://doi.org/10.17815/jlsrf-7-183

[2] TOP500.org. "November 2023," TOP500, Accessed: Mar. 11, 2024. [Online]. Available: https://www.top500.org/lists/top500/2023/11/

[3] J. C. Browne *et al.*, "Comprehensive, open-source resource usage measurement and analysis for HPC systems," *Concurr. Comput.*, vol. 26, no. 13, pp. 2191–2209, Sep. 10, 2014, https://doi.org/10.1002/cpe.3245

[4] A. Gómez-Iglesias, C. Rosales, and T. Evans, "Practical monitoring of resource utilization for HPC applications," in *Proc. XSEDE16 Conf. Divers. Big Data Sci. Scale* in XSEDE16, Miami, FL, USA, Jul. 17–21, 2016, pp. 1–8, https://doi.org/10.1145/2949550.2949643

[5] D. Nikitenko, K. Stefanov, S. Zhumatiy, V. Voevodin, A. Teplov, and P. Shvets, "System monitoring-based holistic resource utilization analysis for every user of a large HPC center," in *Algorithms Archit. Parallel Process.* in LNCS, J. Carretero *et al.*, Eds. Granada, Spain, Dec. 14–16, 2016, vol. 10049, pp. 305–318, https://doi.org/10.1007/978-3-319-49956-7_24

[6] F. Wang, S. Oral, S. Sen, and N. Imam, "Learning from five-year resource-utilization data of Titan system," in *2019 IEEE Int. Conf. Clust. Comput. (CLUSTER)*, Albuquerque, NM, USA, Sep. 23–26, 2019, pp. 1–6, https://doi.org/10.1109/CLUSTER.2019.8891001

[7] A. Netti, W. Shin, M. Ott, T. Wilde, and N. Bates, "A conceptual framework for HPC operational data analytics," in *2021 IEEE Int. Conf. Clust. Comput. (CLUSTER)*, Portland, OR, USA, Sep. 7–10, 2021, pp. 596–603, https://doi.org/10.1109/Cluster48925.2021.00086

[8] G. Michelogiannakis *et al.*, "A case for intra-rack resource disaggregation in HPC," *ACM Trans. Archit. Code Optim.*, vol. 19, no. 2, pp. 1–26, Mar. 7, 2022, https://doi.org/10.1145/3514245

[9] J. Li, G. Michelogiannakis, B. Cook, D. Cooray, and Y. Chen, "Analyzing resource utilization in an HPC system: A case study of NERSC's Perlmutter," in *High Perform. Comput.* in LNCS, A. Bhatele, J. Hammond, M. Baboulin, and C. Kruse, Eds. Hamburg, Germany, May 21–25, 2023, vol. 13948, pp. 297–316, https://doi.org/10.1007/978-3-031-32041-5_16

[10] A. Turner and S. McIntosh-Smith, "A survey of application memory usage on a national supercomputer: An analysis of memory requirements on ARCHER," in *High Perform. Comput. Syst. Perform. Model. Benchmarking Simul.* in LNCS, S. Jarvis, S. Wright, and S. Hammond, Eds. Denver, CO, USA, Nov. 13, 2017, vol. 10724, pp. 250–260, https://doi.org/10.1007/978-3-319-72971-8_13

[11] D. Zivanovic *et al.*, "Main memory in HPC: Do we need more or could we live with less?" *ACM Trans. Archit. Code Optim.*, vol. 14, no. 1, pp. 1–26, Mar. 6, 2017, https://doi.org/10.1145/3023362

[12] G. Panwar *et al.*, "Quantifying memory underutilization in HPC systems and using it to improve performance via architecture support," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture* in MICRO '52, Columbus, OH, USA, Oct. 12–16, 2019, pp. 821–835, https://doi.org/10.1145/3352460.3358267

[13] I. Peng, I. Karlin, M. Gokhale, K. Shoga, M. Legendre, and T. Gamblin, "A holistic view of memory utilization on HPC systems: Current and future trends," in *Proc. Int. Symp. Mem. Syst.* in MEMSYS '21, Washington, DC, USA, Sep. 27–30, 2021, pp. 1–11, https://doi.org/10.1145/3488423.3519336

[14] NERSC. "Perlmutter," Accessed: Jan. 9, 2024. [Online]. Available: https://www.nersc.gov/systems/perlmutter/

[15] G. Michelogiannakis *et al.*, "Efficient intra-rack resource disaggregation for HPC using co-packaged DWDM photonics," in *2023 IEEE Int. Conf. Clust. Comput. (CLUSTER)*, Santa Fe, NM, USA, Oct. 31–Nov. 3, 2023, pp. 158–172, https://doi.org/10.1109/CLUSTER52292.2023.00021

[16] V. R. Kommareddy, C. Hughes, S. Hammond, and A. Awad, "Investigating fairness in disaggregated non-volatile memories," in *2019 IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Miami, FL, USA, Jul. 15–17, 2019, pp. 104–110, https://doi.org/10.1109/ISVLSI.2019.00028

[17] I. Peng, R. Pearce, and M. Gokhale, "On the memory underutilization: Exploring disaggregated memory on HPC systems," in *2020 IEEE 32nd Int. Symp. Comput. Archit. High Perform. Comput. (SBAC-PAD)*, Porto, Portugal, Sep. 9–11, 2020, pp. 183–190, https://doi.org/10.1109/SBAC-PAD49847.2020.00034

[18] J. Wahlgren, G. Schieffer, M. Gokhale, and I. Peng, "A quantitative approach for adopting disaggregated memory in HPC systems," in *SC '23 Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, Denver, CO, USA, Nov. 12–17, 2023, pp. 1–14, https://doi.org/10.1145/3581784.3607108

[19] F. V. Zacarias, P. Carpenter, and V. Petrucci, "Improving HPC system throughput and response time using memory disaggregation," in *2021 IEEE 27th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Beijing, China, Dec. 14–16, 2021, pp. 283–290, https://doi.org/10.1109/ICPADS53394.2021.00041

[20] F. Zacarias, P. Carpenter, and V. Petrucci, "Dynamic memory provisioning on disaggregated HPC systems," in *SC-W '23 Proc. SC '23 Workshops Int. Conf. High Perform. Comput. Netw. Storage Anal.* in SC-W '23, Denver, CO, USA, Nov. 12–17, 2023, pp. 973–982, https://doi.org/10.1145/3624062.3624174

[21] M. Copik, M. Chrapek, L. Schmid, A. Calotoiu, and T. Hoefler, "Software resource disaggregation for HPC with serverless computing," in *2024 IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, San Francisco, CA, USA, May 27–31, 2024, pp. 139–156, https://doi.org/10.1109/IPDPS57955.2024.00021

[22] J. Li *et al.*, "Job scheduling in high performance computing systems with disaggregated memory resources," in *2024 IEEE Int. Conf. Clust. Comput. (CLUSTER)*, Kobe, Japan, Sep. 24–27, 2024, pp. 297–309, https://doi.org/10.1109/CLUSTER59578.2024.00033

[23] S. Blagodurov and A. Fedorova, "Towards the contention aware scheduling in HPC cluster environment," in *J. Phys. Conf. Ser.*, Vancouver, BC, Canada, May 1–3, 2012, vol. 385, p. 012 010, https://doi.org/10.1088/1742-6596/385/1/012010

[24] C. Galleguillos, A. Sîrbu, Z. Kiziltan, O. Babaoglu, A. Borghesi, and T. Bridi, "Data-driven job dispatching in HPC systems," in *Mach. Learn. Optim. Big Data* in LNCS, G. Nicosia, P. Pardalos, G. Giuffrida, and R. Umeton, Eds. Volterra, Italy, Sep. 14–17, 2017, vol. 10710, pp. 449–461, https://doi.org/10.1007/978-3-319-72926-8_37

[25] A. Netti, C. Galleguillos, Z. Kiziltan, A. Sîrbu, and O. Babaoglu, "Heterogeneity-aware resource allocation in HPC systems," in *High Perform. Comput.* in LNCS, R. Yokota, M. Weiland, D. Keyes, and C. Trinitis, Eds. Frankfurt, Germany, Jun. 24–28, 2018, vol. 10876, pp. 3–21, https://doi.org/10.1007/978-3-319-92040-5_1

[26] Q. Xiong, E. Ates, M. C. Herbordt, and A. K. Coskun, "Tangram: Colocating HPC applications with oversubscription," in *2018 IEEE High Perform. Extrem. Comput. Conf. (HPEC)*, Waltham, MA, USA, Sep. 25–27, 2018, pp. 1–7, https://doi.org/10.1109/HPEC.2018.8547644

[27] F. V. Zacarias, V. Petrucci, R. Nishtala, P. Carpenter, and D. Mosse, "Intelligent colocation of workloads for enhanced server efficiency," in *2019 31st Int. Symp. Comput. Archit. High Perform. Comput. (SBAC-PAD)*, Campo Grande, Brazil, Oct. 15–18, 2019, pp. 120–127, https://doi.org/10.1109/SBAC-PAD.2019.00030

[28] A. V. Goponenko, R. Izadpanah, J. M. Brandt, and D. Dechev, "Towards workload-adaptive scheduling for HPC clusters," in *2020 IEEE Int. Conf. Clust. Comput. (CLUSTER)*, Kobe, Japan, Sep. 14–17, 2020, pp. 449–453, https://doi.org/10.1109/CLUSTER49012.2020.00064

[29] F. V. Zacarias, V. Petrucci, R. Nishtala, P. Carpenter, and D. Mossé, "Intelligent colocation of HPC workloads," *J. Parallel Distrib. Comput.*, vol. 151, pp. 125–137, May 2021, https://doi.org/10.1016/j.jpdc.2021.02.010

[30] A. Tzenetopoulos, D. Masouros, S. Xydis, and D. Soudris, "Interference-aware workload placement for improving latency distribution of converged HPC/big data cloud infrastructures," in *Embed. Comput. Syst. Archit. Model. Simul.* in LNCS, A. Orailoglu, M. Jung, and M. Reichenbach, Eds. Virtual Event, Jul. 4–8, 2021, vol. 13227, pp. 108–123, https://doi.org/10.1007/978-3-031-04580-6_8

[31] S. Xue *et al.*, "Kronos: Towards bus contention-aware job scheduling in warehouse scale computers," *Front. Comput. Sci.*, vol. 17, no. 1, p. 171 101, Aug. 8, 2022, https://doi.org/10.1007/s11704-021-0418-5

[32] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," *SIGPLAN Not.*, vol. 45, no. 3, pp. 129–142, Mar. 13, 2010, https://doi.org/10.1145/1735971.1736036

[33] D. Xu, C. Wu, and P.-C. Yew, "On mitigating memory bandwidth contention through bandwidth-aware scheduling," in *Proc. 19th Int. Conf. Parallel Archit. Compil. Tech.*, Vienna, Austria, Sep. 11–15, 2010, pp. 237–248, https://doi.org/10.1145/1854273.1854306

[34] A. D. Breslow *et al.*, "The case for colocation of high performance computing workloads," *Concurr. Comput.*, vol. 28, no. 2, pp. 232–251, Dec. 10, 2013, https://doi.org/10.1002/cpe.3187

[35] D. Dauwe *et al.*, "HPC node performance and energy modeling with the co-location of applications," *J. Supercomput.*, vol. 72, no. 12, pp. 4771–4809, Jun. 24, 2016, https://doi.org/10.1007/s11227-016-1783-y

[36] A. Kuity and S. K. Peddoju, "cHPCe: Data locality and memory bandwidth contention-aware containerized HPC," in *ICDCN '23 Proc. 24th Int. Conf. Distrib. Comput. Netw.*, Kharagpur, India, Jan. 4–7, 2023, pp. 160–166, https://doi.org/10.1145/3571306.3571402

[37] Forschungszentrum Jülich. "JUWELS: Jülich Wizard for European Leadership Science," Jülich Supercomputing Centre (JSC), Accessed: Nov. 16, 2023. [Online]. Available: https://www.fz-juelich.de/en/ias/jsc/systems/supercomputers/juwels

[38] E. Suarez, N. Eicker, and T. Lippert, "Supercomputer evolution at JSC," in *NIC Symp. 2018* in Publication Series of the John von Neumann Institute for Computing (NIC) NIC Series, K. Binder, M. Müller, and A. Trautmann, Eds. Jülich, Germany, Feb. 22–23, 2018, vol. 49, pp. 1–12, ISBN: 978-3-95806-285-6. [Online]. Available: http://hdl.handle.net/2128/17546

[39] E. Suarez, N. Eicker, and T. Lippert, "Modular supercomputing architecture: From idea to production," in *Contemporary High Performance Computing: From Petascale toward Exascale* in Chapman & Hall/CRC Computational Science Series, J. S. Vetter, Ed., red. by Sahni, Sartaj, 1st ed., vol. 3, 3 vols., Boca Raton, FL, USA: CRC Press, May 16, 2019, pp. 223–255, https://doi.org/10.1201/9781351036863-9

[40] E. Suarez *et al.*, "Modular supercomputing architecture: A success story of European R&D," ETP4HPC, White Paper, May 11, 2022, https://doi.org/10.5281/zenodo.6508393

[41] M. Riedel *et al.*, "Practice and experience in using parallel and scalable machine learning with heterogenous modular supercomputing architectures," in *2021 IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, Portland, OR, USA, Jun. 17–21, 2021, pp. 76–85, https://doi.org/10.1109/IPDPSW52791.2021.00019

[42] N. Eicker and T. Lippert, "An accelerated cluster-architecture for the exascale," *PARS-Mitteilungen*, vol. 28, no. 1, pp. 110–119, Oct. 2011, https://doi.org/10.1007/BF03341990

[43] D. Alvarez Mallon, N. Eicker, M. E. Innocenti, G. Lapenta, T. Lippert, and E. Suarez, "On the scalability of the clusters-booster concept: A critical assessment of the DEEP architecture," in *Proc. Futur. HPC Syst. Chall. Power-Constrained Perform.* in FutureHPC '12, Venezia, Italy, Jun. 25, 2012, pp. 1–10, https://doi.org/10.1145/2322156.2322159

[44] A. Kreuzer, N. Eicker, J. Amaya, and E. Suarez, "Application performance on a cluster-booster system," in *2018 IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, Vancouver, BC, Canada, May 21–25, 2018, pp. 69–78, https://doi.org/10.1109/IPDPSW.2018.00019

[45] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux utility for resource management," in *Job Sched. Strateg. Parallel Process.* in LNCS, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Seattle, WA, USA, Jun. 24, 2003, vol. 2862, pp. 44–60, https://doi.org/10.1007/10968987_3

[46] M. A. Jette and T. Wickberg, "Architecture of the Slurm workload manager," in *Job Sched. Strateg. Parallel Process.* in LNCS, D. Klusáček, J. Corbalán, and G. P. Rodrigo, Eds. St. Petersburg, FL, USA, May 19, 2023, vol. 14283, pp. 3–23, https://doi.org/10.1007/978-3-031-43943-8_1

[47] SchedMD. "Slurm workload manager," Accessed: Dec. 12, 2023. [Online]. Available: https://slurm.schedmd.com/

[48] Y. Müller, F. S. M. Guimarães, C. Karbach, and W. Frings, *LLview*, version v2.2.3-base, Zenodo, Feb. 13, 2024, https://doi.org/10.5281/zenodo.10221407

[49] "LLview," Accessed: Nov. 28, 2023. [Online]. Available: https://llview.fz-juelich.de

[50] Prometheus Authors. "Prometheus - Monitoring system & time series database," Accessed: Mar. 20, 2024. [Online]. Available: https://prometheus.io/

[51] NVIDIA. "NVIDIA Management Library (NVML)," NVIDIA Developer, Accessed: Mar. 4, 2024. [Online]. Available: https://developer.nvidia.com/nvidia-management-library-nvml

[52] The pandas development team, *Pandas-dev/pandas: Pandas*, version 2.2.0, Zenodo, Jan. 20, 2024, https://doi.org/10.5281/zenodo.3509134

[53] W. McKinney, "Data structures for statistical computing in python," in *Proc. 9th Python Sci. Conf.*, S. van der Walt and J. Millman, Eds. Austin, TX, USA, Jun. 28–Jul. 3, 2010, pp. 56–61, https://doi.org/10.25080/Majora-92bf1922-00a

[54] NVIDIA. "NVIDIA DCGM," NVIDIA Developer, Accessed: Mar. 4, 2024. [Online]. Available: https://developer.nvidia.com/dcgm