Improving Computing Efficiency and Reducing Carbon Footprint for Turing Cluster

An Interactive Qualifying Project Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

Qixing Xue
Advised by Fabricio Murai and Ermal Toto

March 19, 2024

Abstract

With the increasing popularity of large language models (LLM) and its great potential of renovating the way we work, now, more than ever, high performance computing (HPC) is paramount for training accurate machine learning models and creating scientific breakthroughs in numerous subjects. However, due to arising concerns of its carbon footprint and ecological impact, user education of proper cluster usage is vital for improving energy efficiency, while administrators also need a tool to gain understanding of utilization status and identify problems, so to better manage the cluster and guide its users. In order to achieve this, it can be helpful to increase the observability of computing resource utilization that is previously impacted by workload managers like SLURM isolating compute nodes from login nodes and make it harder for users to interact with the former ones. This increased observability in turn reveals the problems and can raise the awareness of underutilization. In this project, we implement an automated tool that collects statistics and GPU driver readings for readings of processes under job allocations and generates actionable reports with user-friendly explanations of potential problems in job submissions and their possible solutions. The tool can both be started by regular users along with their job allocations without administrator involvement or be deployed by administrator as daemons to sample the cluster at tunable frequency, with both data collection and problem identification components being highly extensible. By running on the Turing HPC cluster in WPI for roughly half a year, it has helped users to better understand running status of their jobs and has aided administrators in identifying job problems across multiple dimensions, such as user, resource, and time period.

Acknowledgements

I want to thank James Kingsley in Academic & Research Computing group for initially proposing the topic, providing the implementation with needed resources and information, and suggesting improvements in user experience, Reily Siegel in Academic & Research Computing group for providing input in improving report format and infrastructures implemented for exporting power usage data. This research was performed using computational resources supported by the Academic & Research Computing group at Worcester Polytechnic Institute.

Contents

Part I.

Introduction

1. Overview

High performance computing enables scientific and engineering breakthroughs [25] by providing computationally intense domain-specific software with powerful computing resources [12] that entails at least hundreds to thousands of CPU cores, dozens to hundreds of gigabytes of RAM, as well as hardware accelerators like NVIDIA A100 that could each perform around 20 trillion float operations (FLOPs) per second [30]. For this capability, it has been used in multiple natural science fields including physics, astronomy, chemistry, geology and material sciences in early years [11] and is recently being applied onto artificial intelligence, especially deep learning, backed by advancements in both theory and hardware [31]. The advantage of high compute power naturally comes with the cost of high energy consumption, which eventually leads to high carbon footprint [18] that will keep growing when without intervention [10], contributing to global warming and climate change [7]. Together with the pollution from handling e-wastes [2] from hardware upgrades, it brings risks to health, water, food, and meanwhile possibly hurt economy development [21]. The power consumption and corresponding carbon footprint of HPC clusters are being further driven up by the increasing adoption of high performance computing by machine learning community for the difference in performance and price of compute cards in contrast to regular customer-grade GPUs. However, its methodology and efficiency brings concerns [27], due to the nature of model training being experimental and the practice of frequently discarding results of unsatisfactory accuracies even though significant amount of power is consumed on producing them.

As of October 12, 2023, WPI operates a Turing cluster consisting a total of 46 compute nodes, 3492 CPU cores, 40.5 TB of RAM and 60 GPU cards. Based on hardware power usage data collected from Jan. 18, 2024 to Feb. 7 2024 that does not include facility power usage like those for cooling, Figure 1.1 shows that the usage roughly follows normal distribution, and, on average, fluctuates by roughly 1 kW throughout the day and falls in a range of 36.365 to 37.59 kW, with peak usage occurring at night, as in Figure 1.2. Using this data, we estimate the carbon footprint from purely power draws of hardware. As shown in Table 1.1, the 95% confidence range[1] of power usage falls between 36.88 and 37.16 kW which, per year, corresponds to 323.13-325.56 MWh of power consumption, or 156.72-157.90 tons of CO_2 emission per year – following the 2022 US EPA data that natural gas, has an average carbon emission of 0.000485 ton of CO_2 per kWh. Natural gas is chosen for calculation considering that it is known to be used by WPI prior to 2023 and is one of the major energy source being used by New England, while also being the source with largest carbon footprint as of February 22, 2024, as shown in Figures 1.3 and 1.4 fetched from ISO New England Dashboard[2] on

[1] $\bar{x} = 37.02611$, $s = 2.1809$, $n = 950$, with sampling frequency of 30 minutes.

[2] `https://www.iso-ne.com/isoexpress/`

February 22 at 8PM. When combined with power consumption from cooling, using the estimation that it takes roughly 30% of electricity needed to power the computational resources [35], the yearly power consumption will be 461.61-465.09 kWh, and therefore corresponds to a carbon footprint of 223.89-225.57 tons of CO_2 per year. Using the estimation that a 3.69kWh LMO–NMC battery have a carbon footprint of 216.2 kg CO_2/kWh in the production phase, 94.2 kg CO_2 eq/kWh in the use phase, and -17.18 kg CO_2 eq/kWh in the recycling phase [17], the aforementioned carbon emission equivalents to roughly 200 life cycles of electric vehicle batteries from raw material to production to recycle. However, as gross measurement of power draw does not differentiate between energy required for the computation and those that could possibly be saved, energy efficiency, as defined by Green500[3] as number of FLOPs per watt, better measures how well energy is being utilized, and optimizing it can reduce environmental implications without negatively impacting efficiency.

	Power Usage (kW)	**Consumption** (MWh/year)	**Equivalent Carbon Emission** (tCO_2/year)
Min	30.5	267.18	129.5823
25%	35.5	310.98	150.8253
95% CI Lower	36.887	323.134	156.7199
50%	37.1	324.996	157.6231
95% CI Higher	37.165	325.563	157.8983
75%	38.5	337.26	163.5711
Max	42	367.92	178.4412

Table 1.1.: Different percentiles and 95% confidence range values of power usage in collection period, converted to yearly power consumption and equivalent carbon emission, assuming natural gas is used, and follows 2022 US EPA data that 0.000485 ton of CO_2 is emitted per kWh generated.

Unlike working on a single shared node, users do not have direct access to nodes performing actual computing, or compute nodes. Instead, they submit a request to workload manager, indicating time and computing resources required, and wait for the resource to be available following a job priority considering account used, job size and past usage history. After the requested resources are allocated, the user may send commands remotely to run multiple steps or simply a batch script and perform actual computations on allocated nodes. SLURM [33] is a workload manager widely used in HPC clusters that allocates each job a set of resources, including CPU cores, RAM, licenses, and hardware accelerators like GPU, for programs to run on one or more nodes for limited amount of time, with which multiple commands are executed as steps, following the specification

[3] `https://top500.org/lists/green500`

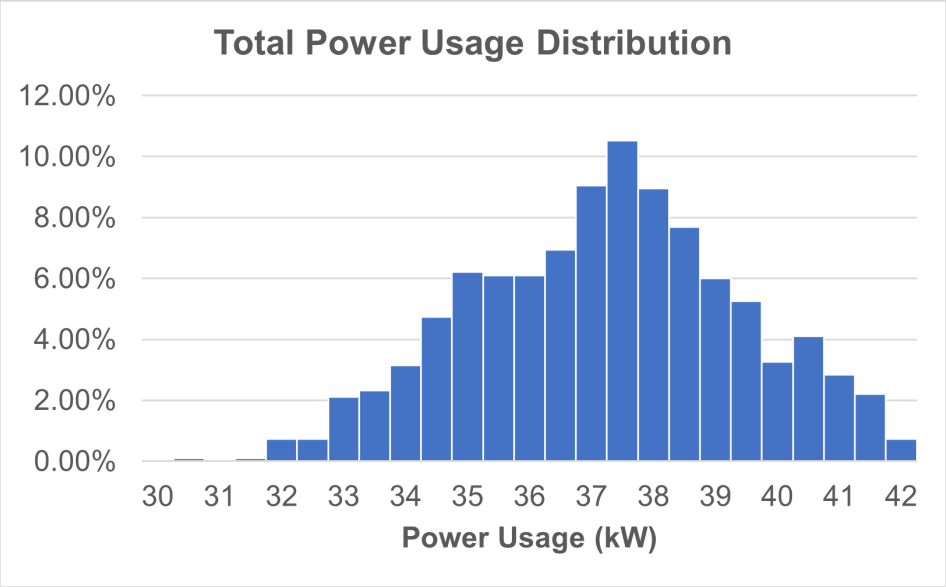


Figure 1.1.: Distribution of power usage data collected from January 18, 2024 to February 7, 2024, excluding facility power consumption like those from cooling, with sampling interval of 30 minutes and displayed in percentage of samples falling in corresponding bins.

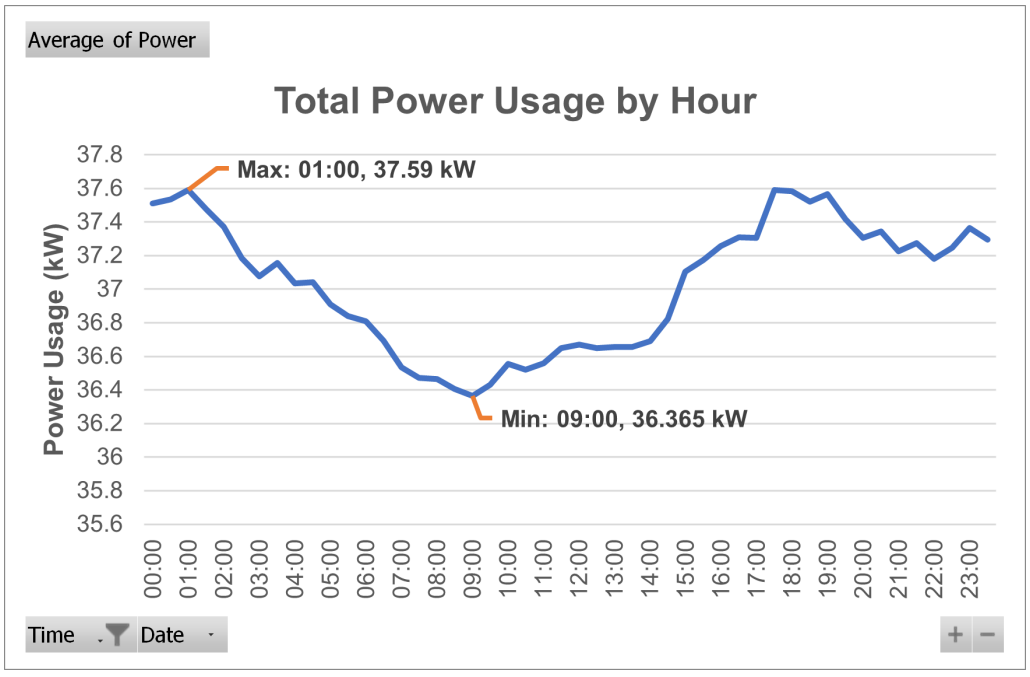


Figure 1.2.: Average power usage by hours plotted using data collected from January 18, 2024 to February 7, 2024, excluding facility power consumption like those from cooling, with sampling interval of 30 minutes, and in local time.

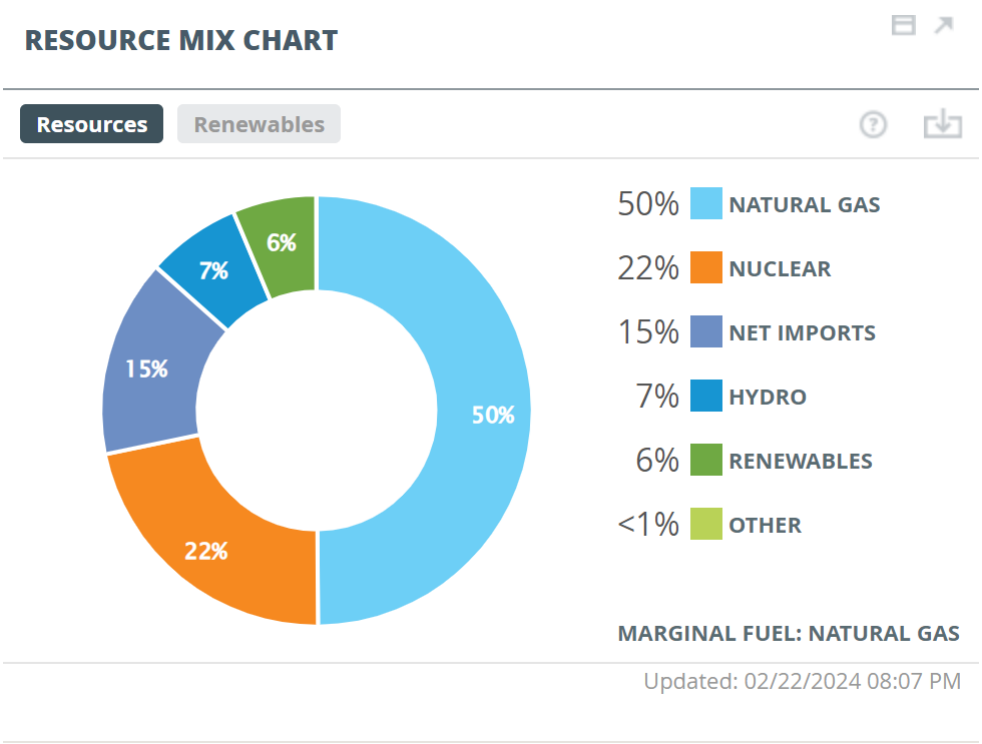


Figure 1.3.: Energy source compositions for electric generation, as provided by ISO New England.

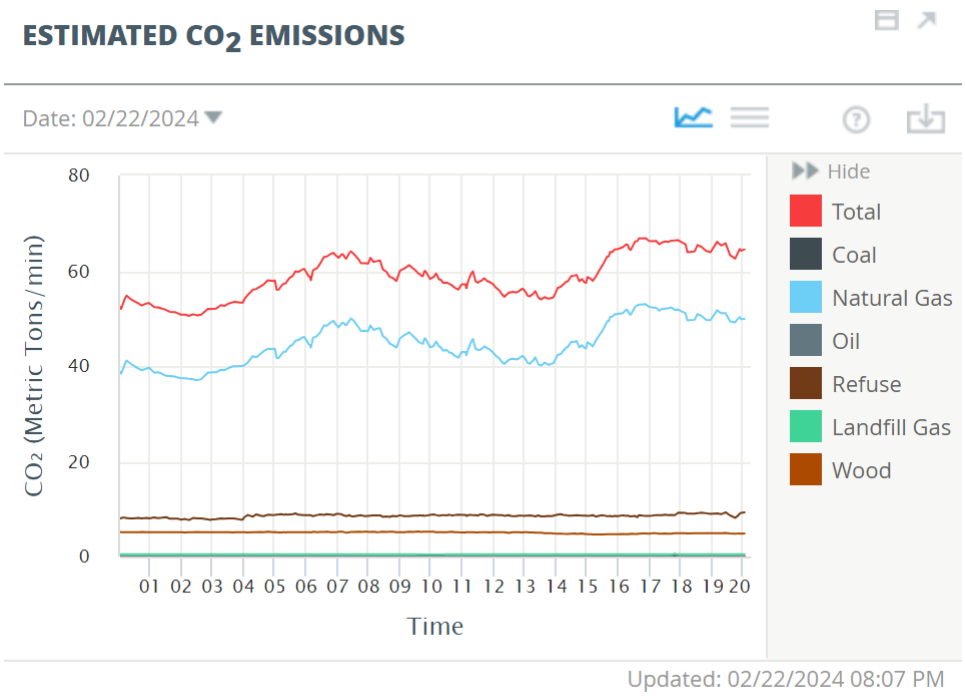


Figure 1.4.: Estimation of CO_2 emissions of different sources for electric generation, as provided by ISO New England.

by the users. Although utilizing SLURM to isolate executions on login nodes and compute nodes ensured the enforcement of resource allocation policy, it also reduced users' observability of their job, and therefore requires extra effort for the scripts to report utilization of resources like CPU cores, memory and GPU, or instead uses resources blindly, overestimating the amount of computational resources needed, or not knowing certain resources is not being utilized at all, wasting resources that could be otherwise allocated to other jobs to use concurrently.

The problem worsens if the user has no reference of running time or performance of program, e.g. from test runs on local machine, possibly due to program being unavailable locally or requires too much memory to run, it may be even harder for them to realize that the job is running under a nonideal condition that might cause it to be computing at a performance slower than running on a laptop. This would also cause power consumption of components whose usage varies less with workload change, e.g., cooling and disk controller, to be amortized by fewer jobs and therefore lowers overall energy utilization.

However, according to our literature review, there is currently no solution fulfilling this demand, as most monitoring tools identified focus on administrators rather than users, reporting to them node health and overall operating status without information at the granularity of individual jobs other than those provided by SLURM accounting interface, that describes job characteristics and flags problematic submissions.

For this reason, the tool proposed here is designed with cluster users in mind: it enables them to start their own instance and collect data along with regular job submission, with minimal environment setup efforts and only minor modifications to job scripts to start and stop tool components, by porting job submission script into provided template.

Based on the collected data, we can easily generate a report that is readable, actionable and persuasive, displaying both summary of identified problems and detailed explanation and common solution, as well as data supporting it, as demonstrated in Appendix A.

While our tool is capable of being used by regular user, it does not surrender its usability by administrators. By providing the functionality of sampling data across nodes in the cluster and generating reports in the style of pivot tables with different aggregation level, as seen in Figure 4.7, it allows them to identify the most prevailing problems intuitively and to communicate with users in a more effective manner by knowing what change to ask for, ultimately bringing the cluster to be better utilized and in higher efficiency.

In order to achieve aforementioned functionality, the proposed tool operates through the collaboration of SLURM and multiple components, with connections depicted in Figure 1.5, data and control flow shown in Figure 1.6 and summarized below:

- **Watcher server (Section 3.1)** establishes a simple server for receiving results from scrapers, which, together with updates from SLURM accounting interface, are used for inserting and updating records in a SQLite database with schema specified in Section 3.2. Involving this component reliefs the need of setting up a full database server, making the tool more portable and easy to set up by cluster users. In addition, it circumvents the need of storing database file in a network location, which helps speed up file access when the tool has a dedicated machine

to run on, as in the case of being used by administrators.

- **Scraper (Section 3.3)** runs for certain number of rounds each time it is invoked, with constant timing interval between each round, during which various resource usage data of processes are fetched and mapped onto job steps that would be sent back to the watcher server at the end of execution. It may also use external data source like the readings collected by samplers of Bright Cluster Manager in case the device driver being inaccessible, for example when being used by administrators and executed under the resource constraints imposed by SLURM job allocation.
- **Scraper Distributor (Section 3.4)** distributes scrapers onto nodes with concurrency control mechanism in place, following an algorithm that prioritizes the allocation of the scrapers in nodes with more running jobs, but ensures allocation on every node with running jobs is attempted during a distribution round. This component is only required if the use case is to sample jobs across the cluster, as when being used by administrators, while regular users could simply run scrapers within their regular job allocation to monitor the resource usage of their own jobs.
- **Analyzer (Chapter 4)** converts the dataset into a unified base table in the form of time series and performs analyses as queries (Section 4.2) to check for presence of resource underutilization by comparing actual usage against allocated amount, as well as certain inefficiencies, such as low GPU utilization or having large proportion of time waiting for system calls. Furthermore, with the tool designed to be extensible in order to adapt to different cluster setups and use cases, analyses can be easily added or modified, while data needed for the added analyses can also be collected and recorded using the scraping and database migrating framework, as discussed in subsection 4.2.4. It then renders reports for individual users using result rows from the queries, in the format of HTML email (Section 4.1) using a predefined template that provides rich navigation functionality and detailed explanations. In addition to receiving periodically reports characterizing jobs since the last was sent, a website (Section 4.3) is also available for users to look for old reports and view the latest report with higher updating frequency than the interval of sending emails, for example per hour v.s. per week, as well as old reports for their reference. The analyzer also combines all result rows from analyses into a JSON file for them to be displayed as a pivot table of multiple aggregation levels, including user and time period. Using this view, administrators can better understand current usage status and locate prevailing problems, while the capability of reading users' report allows them to better communicate with users for their mutual access to same content.

As an result, a complete data pipeline spanning collection, analysis and reporting is established and automated through this project, and is being deployed on Turing cluster. With collection from early September 2023 to late February 2024, more than 5.5 million measurements are collected, occupying around 0.5 GB of storage space. These data are grouped into 16 periods starting November 1, 2023 and allows users to have

more observability over their job submissions and raised awareness on utilization status. This is shown by receiving emails inquiring about fixes to problems identified, as well as certain trend in pivot table view that, for certain job family, it started as having problem for all jobs in the family but no longer had it starting certain time period. By putting effort into resolving these problems, users spend less time waiting in queue, since now they can reduce the amount of resources specified in the request without hurting performance, and hence obtain results faster, by better utilizing allocated resources. This helps ensure users get what they actually need and actually use what they get. Meanwhile, it enables administrators to understand usage status of the cluster in more dimensions, including by user, by resource, and by comparing across different times. This allows them to locate problem more easily and communicate more effectively, by having job characteristics easily obtainable and aggregated following specific needs puts them into a more vivid context.

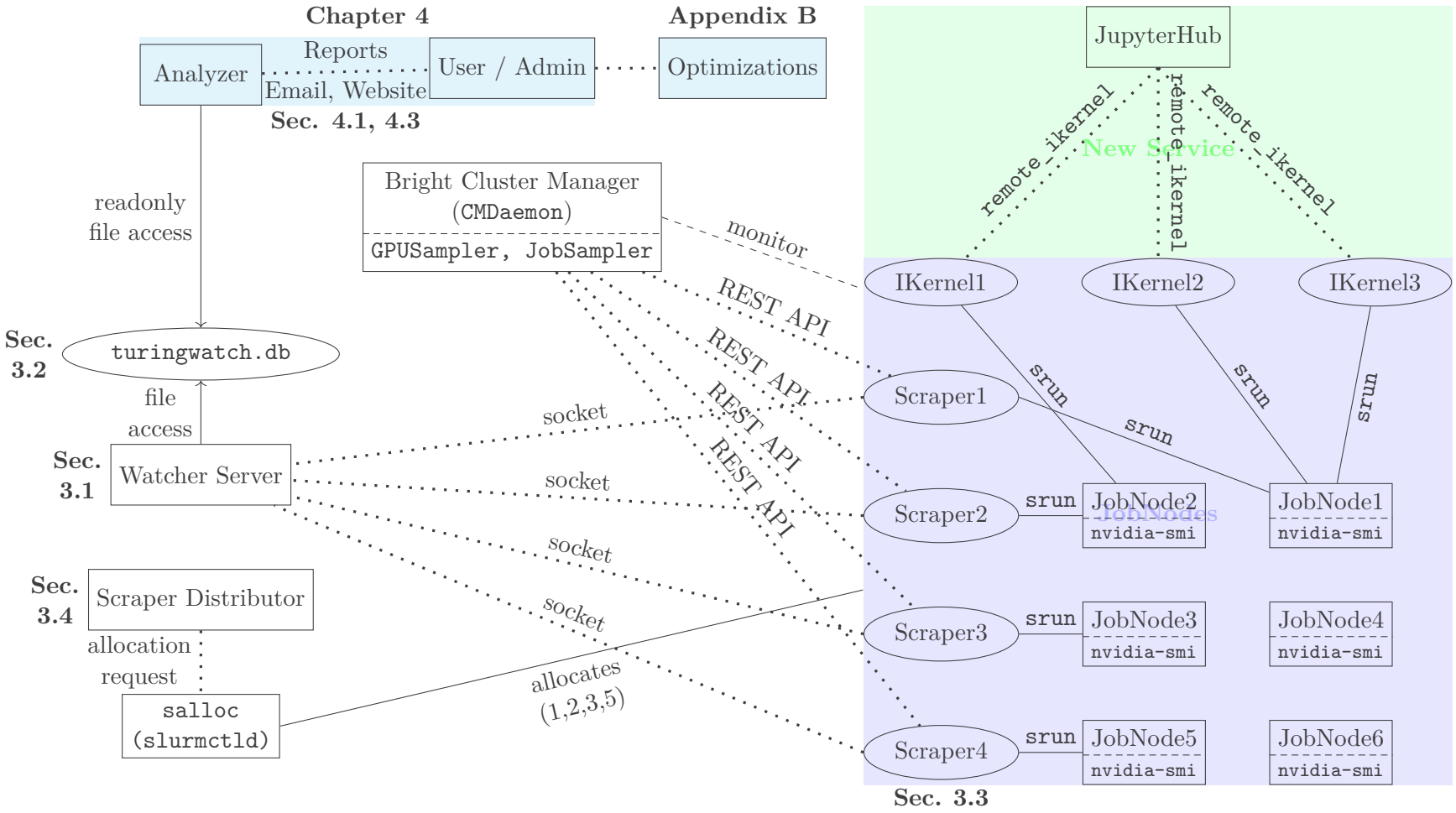


Figure 1.5.: Collaboration graph of workers in watcher, with example of 4 concurrent scrapers

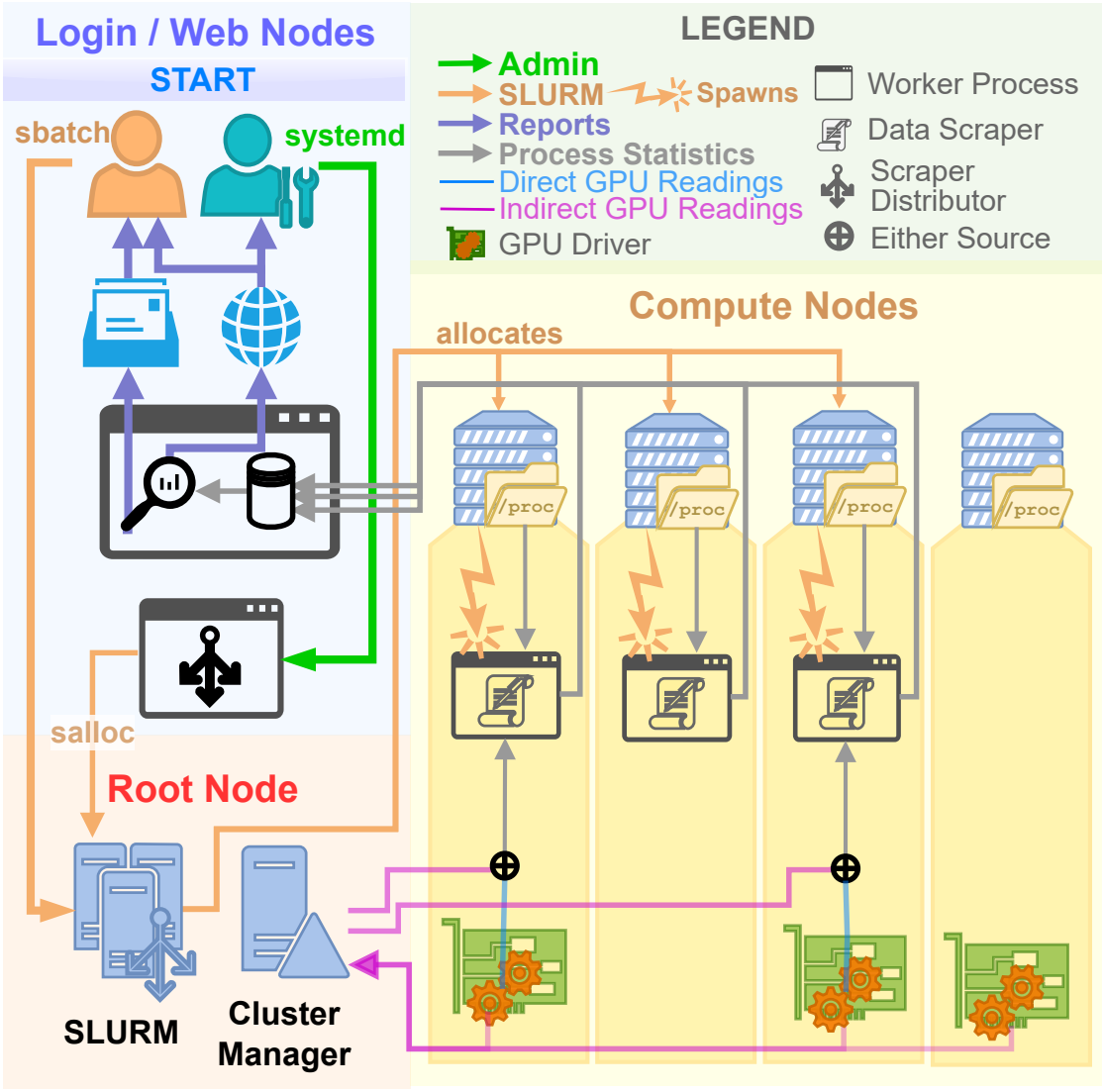


Figure 1.6.: Diagram demonstrating flow of commands and data among components. The scraper is started by users along with job submission using `sbatch`, or by administrators with `systemd`, in which case scrapers will be distributed through allocating jobs. Either way, the scrapers are spawned on compute nodes and read data from `/proc` and GPU driver readings, which comes either directly from driver communications or indirectly from data collected by the cluster manager, and send them back to the watcher server. The watcher server periodically invokes analyzer, which produce reports that can be received through email and viewed on a website by users and administrators.

2. Related Work

In an effort to recognize and reduce the environmental impact of HPC clusters, numerous studies were performed with different concentrations in multiple subjects, including artificial intelligence, green computing, data analytics, and HPC infrastructures. By analyzing related works, it helps in understanding current status of HPC clusters from both environment and operational aspects, and identify the need and difficulties of HPC cluster operations in terms of improving computing efficiency and reducing carbon footprint, ultimately contributing to making the project useful in practice.

2.1. Environmental Impact

Given that power consumption and environmental implication being one major background of this project, reviewing existing literatures help better understand current difficulties and state-of-art. As defined by Hogan [13], data centers host computing infrastructures like servers and nodes of HPC clusters as stacked racks, as well as components supporting them, including storage, cooling, and backup power. Lannelongue, Grealey, and Inouye indicate [18] that data centers produce 100 megatonnes of CO_2 emission yearly and contributes substantially to global warming and hence climate change [7], while Freitag et al. suggest [10] that the amount is going to increase without active intervention, as computing resources are less likely to be saturated. In addition, upgrading hardware for state-of-art performance also possibly creates e-wastes, which creates multiple forms of pollutants as suggested [2] by Ankit et al. The combined effect of climate change and pollution brings hazard to humanity in aspects including health, water, food [2] and possibly economy [21]. The article by Filiposka, Mishev, and Juiz [9] also mentions that reducing power consumption not only saves constructional and operational costs, but also increases reliability and availability of the system.

Although the energy efficiency data of top high performance computers is regularly published on Top500[1] and Green500, they are coarsely aggregated and make no connection to workload, and therefore are less useful for evaluating environment impact of clusters. In order to measure the environment impact and effectiveness in a finer granularity, more than 200 models have been proposed as of 2016 [8], covering different aspects of a cluster, including different job categories, major pieces of server hardware, cooling, and operating system. Despite that large number of models are available and that energy efficiency was brought into attention as early as 2008 by Kamil, Shalf, and Strohmaier [16], there is few identified dataset for understanding environmental implication of HPC clusters that is public available and valid across time and cluster configuration until deep

[1] `https://top500.org/lists/top500`

learning gains its popularity through the development of deep convolutional network and utilization of hardware accelerators, as marked by the publishing of *AlexNet* in 2015 that is described by Tan and Lim as landmark of AI Renaissance, for it being the only ANN-based work for object recognition in its publishing year while was used in almost all works on this topic in subsequent years. In the benchmark performed by Svedin et al. [30], the result indicates that NVIDIA V100, a data center GPU released two years earlier than consumer grade GPU NVIDIA RTX 2060 Super, has double the performance of latter in terms of FP32 FLOPs per second, while performance of FP64 FLOPs per second differs by more than 30 times, making training on HPC environment highly attractive.

In the context of machine learning, hyperparameters are a set of values that, unlike model parameters, needs to be set outside of training process to control the architecture and behavior of the model [23], and regulate the flexibility of model and prevent overfitting that causes lose of accuracy while predicting [15]. Passos and Mishra [23] remarks that hyperparameter selection depends on users' expertise and understanding of the model architecture and affects training and inference time significantly, while there are usually hundreds of them in a larger model and the values could sometimes be in float type, further expanding the number of possible values. Schwartz et al. points out [27] that publications frequently focus on accuracy of models rather than efficiency and cost of training models, especially the part of hyperparameter tuning, enabling the result to be "purchased" with higher energy consumption and therefore raising the barrier of participation. Lannelongue et al. [19] suggest advocating findable, accessible, interoperable, and reusable (FAIR) code, data, and models to reduce unnecessary data generation and storage, meanwhile improving implementation efficiency and enlarging project impact.

The environmental impact of this practice is significant, as Strubell, Ganesh, and McCallum estimates [28] that, although a single training of NLP model only produces 39 lbs of CO_2, this number skyrockets to 78,468 lbs when parameter tuning is taken into consideration, which is equivalent to double of the emission from a regular American life. In a more recent commentary, Vries indicates [32] that the training phase of a few language models, namely GPT-3, Gopher, OPT3, and BLOOM, respectively consumed 1,287, 1,066, 324 and 433 MWh of power. It is also mentioned in the same article that power consumption from inference stage is often significant but ignored, as ChatGPT is estimated to be consuming 2.9 Wh per request and 564 MWh per day. It is also mentioned that 60% of energy consumption in AI from 2019 to 2021 could come from inferencing, but is under influence of parameters like retraining rate and energy–accuracy trade-off.

The trend of using AI techniques to learn from dataset also attracted domain-specific applications in plasma physics, cosmology, gravitational wave astrophysics, high energy physics, multi-messenger astrophysics, materials science, and genetic data, which is described by Huerta et al. [14] as convergence of HPC and AI. In their correspondence, Augier et al. highlighted [3] the takeaway in the original article [24] that "scientists should be mindful of their carbon footprint" and suggested raising attention in user education, as choosing the right tool, profiling and parallelizing the code, and using shared cluster and job scheduler in a right manner contributes to higher energy efficiency. In

addition, the survey [26] performed by Schlagkamp et al. shows that experience level of HPC users positively correlates with their satisfactory level of the service.

2.2. Monitoring in the context of HPC

The interview with multiple cluster administrator reported by Allcock et al. [1] expressed the desired functionality of monitoring tools, namely being able to show current status and generate reports at any desired level of detail for any given time period, and to locate what could be improved, including identifying patterns that are different from those assumed during initial configuration. In addition, for an administrator it is more important to value users' time and attention than hardware, for their effort pushing the boundary of human knowledge. As part of the conclusion, Allcock et al. suggests forming active community to collaborate on a mature, integrated and optimal solution that is less labor-intensive.

Numerous monitoring tools applicable for HPC environment exist as early as 2012 to monitor problems ranging from service quality like response time to security and Service Level Agreement (SLA) violation, as shown in the survey performed by Benedict [5]. In addition, Benedict also identified serval tools measuring environmental impact of HPC systems, through deploying hardware, software, or combination of both. Multiple recent articles proposed tools with modern technology and enhanced functionality, such as the tool proposed by Sukhija et al. [29] enabled custom rules to be set with Prometheus, a database management system designed for monitoring system and timeseries, whereas the tools proposed by Beneventi et al. [6] and Yoo et al. [34] both involve techniques for analyzing big data. However, many of the automated tools are commonly concerned about node and hardware health rather than job status. The tool OpenXDMoD developed by Palmer et al. [22] uses external data source to flag problematic jobs, such as those with CPU under-usage, with data stored in a database server like MySQL or MariaDB and backend supported by packages including NodeJS, LibreOffice, and Chromium. In addition to deploying monitoring utilities, there is also work performing manual analysis on data obtained from these utilities, as Zhu, Neuwirth, and Lippert [36] performed data analysis with regard to I/O pattern on data collected throughout an entire year, while Liu et al. [20] performed their analysis with more diversed dataset that includes hardware counters and added jobs as a dimension of analysis.

Part II.

Implementation

3. Watcher: Data Collection

The watcher is at the core of this project, as its responsibility of collecting performance measurements of jobs on the cluster enables scattered and single-point data to be aggregated into a representation of job step characteristics in different time slices. By doing this, it helps effectively locate problematic submissions and guide human intervention and further profiling and optimization efforts. With implementing a scraper to fetch process statistics and GPU data, for the data from existing providers like SLURM and Bright Cluster Manager are not collected for the purpose of depicting job characteristics, it naturally raises the need of spawning the scraper onto compute nodes. In addition to manually spawning by administrators, or being spawned along with users' job allocations, the implementation also contains a job-queue aware scraper distributor that dispatches scrapers onto nodes as regular jobs. For adaptability of different infrastructures and use cases, all components implemented are under tunable frequency control so to ensure that the intensity of RPCs like job allocations and queries are not overwhelming, while enough data are collected for the reports to be meaningful. To simplify usage, multiple scripts and `systemd` service file templates are provided for users to use the tool in a single command after simple and guided modifications.

3.1. Interacting with the Database

For the purpose of analyzing job characteristics, a data source that supplies abundant information for purpose needs to be considered. Although SLURM and Bright Cluster Manager available for use in the cluster collect similar metrics as those in out database schema, the data from them may not be helpful in depicting submission characteristics as they are not designed to serve this purpose, in addition to requiring API keys or even administrative privilege to obtain data. Given that main job of SLURM is to allocate resources and measure usage for evaluating job priorities, it enforces constraints mainly through `cgroup` mechanism and does not store regular monitoring data into accounting database. Furthermore, the real-time measuring functionality `sstat` only supports measuring a single job at a time that meanwhile takes nearly one second to complete, for an RPC request to go through multiple SLURM daemons in the hierarchy and further stress the daemons that can already be busy enough handling allocation requests. On the other hand, although the Bright Cluster Manager has daemons on nodes to collect real time metrics, the data are mostly not mapped onto individual job steps and rather a measurement of the node to monitor health status and raise alerts, in addition to having rather long updating interval of two minutes by default, as this type of software is constantly collecting data from all nodes in the cluster and has to limit growing speed of database size.

For these reasons, it will be desirable to collect a database on our own and therefore a database will be needed. Considering the need of having the application compatible with running at privilege of regular user, SQLite is chosen as the database management software used for maintaining data collected from various sources. With the advantage of portability and no SQL server installation required comes with the drawback that the only way to synchronize among connections is locking file pages when accessed across network file system. Therefore, a watcher server is implemented to hold exclusive access to the database file and serialize queries on it, using the data sent by scrapers through sockets in a custom stream format.

Although being the central component in collaboration, the implementation of watcher server is relatively straightforward, which simply accepts incoming socket connections and stage received measurement data in a message queue. In each periodical update, measurements staged before the end of last insertion period are inserted along with the update of `jobinfo` table that imports latest job accounting information, after which analyzer is invoked and have reports generated. By inserting at this specific timing rather than those received before current period, it reduces complexity of locking design and enforcing constraints of foreign key to `jobinfo` table, by avoiding repeatedly querying SLURM accounting API for information of new jobs appearing since last import whose process statistics are scraped and reported at the last minute before staging insertion is finalized. However, it can be worthwhile to optimize this insertion timing to be more responsive.

3.2. Database Design

As demonstrated in Figure 3.1, the database schema is designed in a relatively straightforward way that stores metric readings from data sources in unified units and match them onto originating job steps. To accommodate the case that a job step using more than one GPU on a node, the GPU measurements are split out into another table and related to the `measurements` table by field `batch`, with each batch consisting of measurements of all GPU in use for the unique measurement entry it is being related to.

With implementing the analyzer in mind, the metrics chosen represents certain characteristics of the execution, but can be added easily with the provided migrating framework as needed. The most indicative metric can be `sys_[u]sec` showing the time spent in kernel space, which measures time waiting certain resources like disk data or memory allocation to be available and is frequently being unhelpful in progressing common HPC computing which prefers larger portion of time to be spent in user space. In addition, the growth rate of other metrics could be established by forming time series, from which the rate and frequency of allocating memory pages can be drawn from `minor_pagefault` and have the behavior be further resembled by the change of resident set size `res`, while the disk and `tty` I/O rate can be derived from change in the values of `dev_in` and `dev_out` in adjacent rows, just to name a few. Jobs having similar inefficient patterns in these metric changes can be of similar problem, so that the optimization could be implemented in a way that is more generally applicable and ultimately promote com-

puting efficiency more efficiently. On the other hand, the GPU metrics serve slightly different purpose, mainly to identify underutilized allocations and rare malfunctioning cases like inadequate cooling, although these usually not be happening in cluster setting given that the server room is being maintained by professionals.

3.3. Scraper

For collecting data, measurements are taken by running scrapers on the target nodes and directly fetch data provided in `/proc` directory and aggregate using process tree formed with parent process ID information, with `slurmstepd` processes mapping process subtrees to job steps. The scraper runs in a relatively high frequency and performance but terminates quickly, rather than keep running for indefinite amount of time, to avoid inflating the database too quickly, although the rate is easily tunable. However, since measuring GPU requires device access not to be denied in `cgroup` policy and therefore naturally requires the card to be allocated by SLURM when sampling the cluster under a job step, indirect measurements from daemons like those from Bright Cluster Manager are utilized when available. Note that user submissions are not impacted by this restriction as they rightfully have access to the card needing to be monitored and only have to properly ensure card access by the scraper either by sharing cards for all tasks or by allowing scraper task to see all cards allocated on the node in newer versions of SLURM, whose instructions are documented in detail in the template submission script. Algorithm 1 sketches the workflow of a scraper run in detail.

3.4. Scraper Distributor

As the scrapers need to be spawned on compute nodes, in addition to starting along with users' jobs or being manually spawned by administrator, it is desirable to have a distributor to sample the cluster so to grow data size in an automated and controllable manner. However, this brings the problem of selecting which nodes to run on for each scraping round, especially when under the constraint of maximum concurrent allocated nodes in a partition or even globally. In this case, in order to learn about characteristics of not only more jobs, but also jobs on different nodes, it will need a distributing mechanism that maximizes the number of job steps measured in one round to know about more jobs, while displaying fairness that makes sure every node available is tried and tries to know about jobs on different nodes. For implementation, in each loop, the distributor reads the number of ongoing jobs on each node in the cluster and sort them and sort them descendingly as the scraping order, after which it selects account to be used in each partition using the algorithm shown in Algorithm 2 and starts attempting allocation onto nodes, with concurrency capped at certain parameter. The node list in each allocation request is composed by combining two queues, old queue and new queue, with combined size of these two queues not exceeding double of the concurrency value, in which new queue consisting of nodes attempted for the first time in this loop, whereas old queue contains nodes failed to be allocated in the last round. When new queue is empty, every

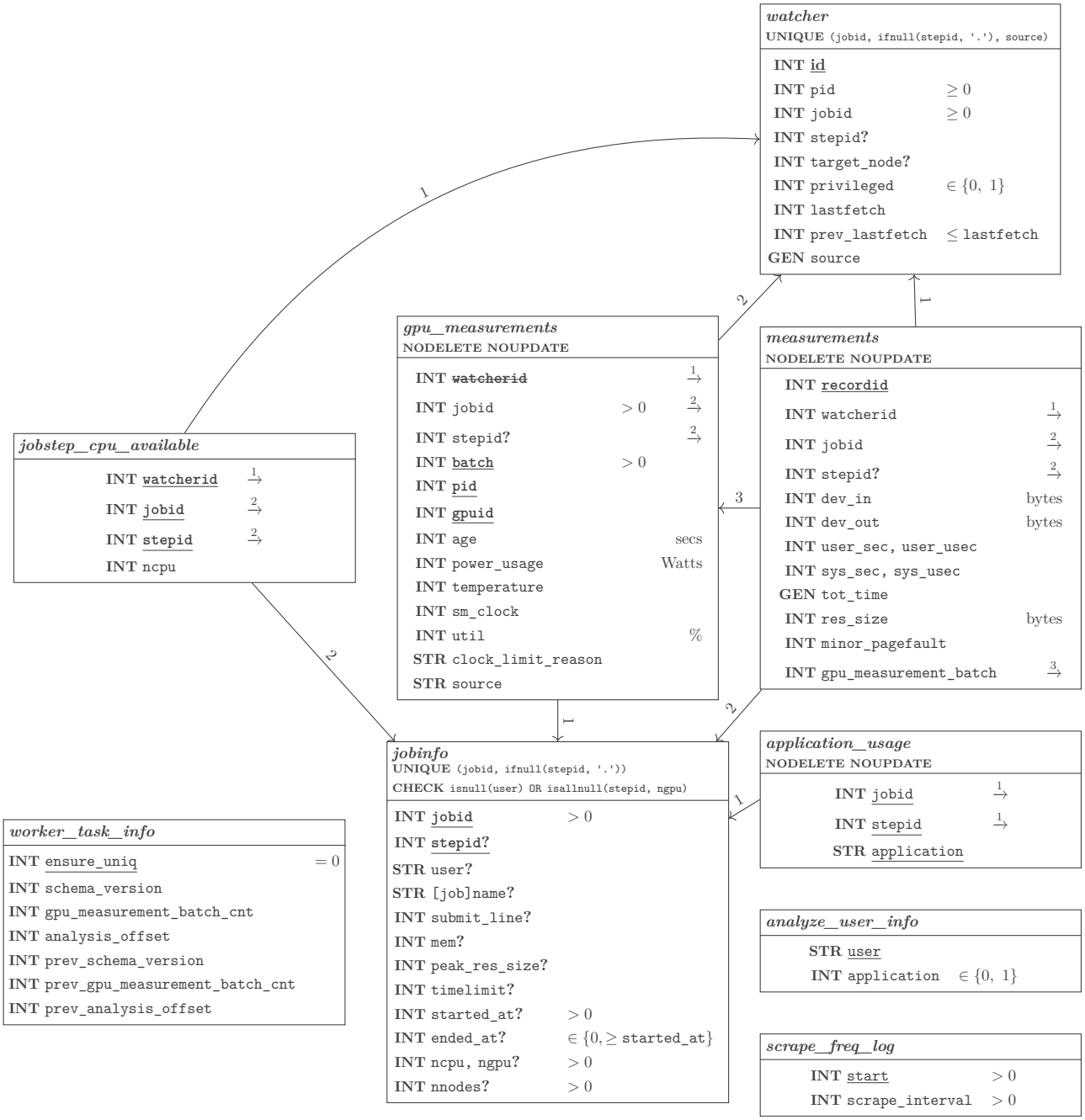


Figure 3.1.: Database schema used by watcher and analyzer.

node that was failed to be allocated in any previous rounds are added into a reattempt queue, in which failed nodes are requested with concurrency limit repeatedly until none of the nodes can be allocated, which terminates a loop. Whenever a node is successfully allocated, it is popped out of the queue, regardless of queue type. To show this more clearly, Figure 3.2 shows diagram of an example round of distribution. By providing a list with size being double of expected concurrency, it ensures concurrency by giving SLURM extra options to choose from. Although it is possible that SLURM prefers nodes provided as alternative, for reasons like having less workload on them, requesting allocation twice for each node in a way that it had both in the category of under heavy and light workload relative to others in the requested list, as well as allocating from reattempt queue until empty, should help relief this problem. However, it is worthwhile to note that the workload situation may have changed at the time of allocation when compared to the time when sorted list is formed, especially if the scrape time for each round is long enough, in which case further development taking changes in number of ongoing jobs on nodes into consideration is desired.

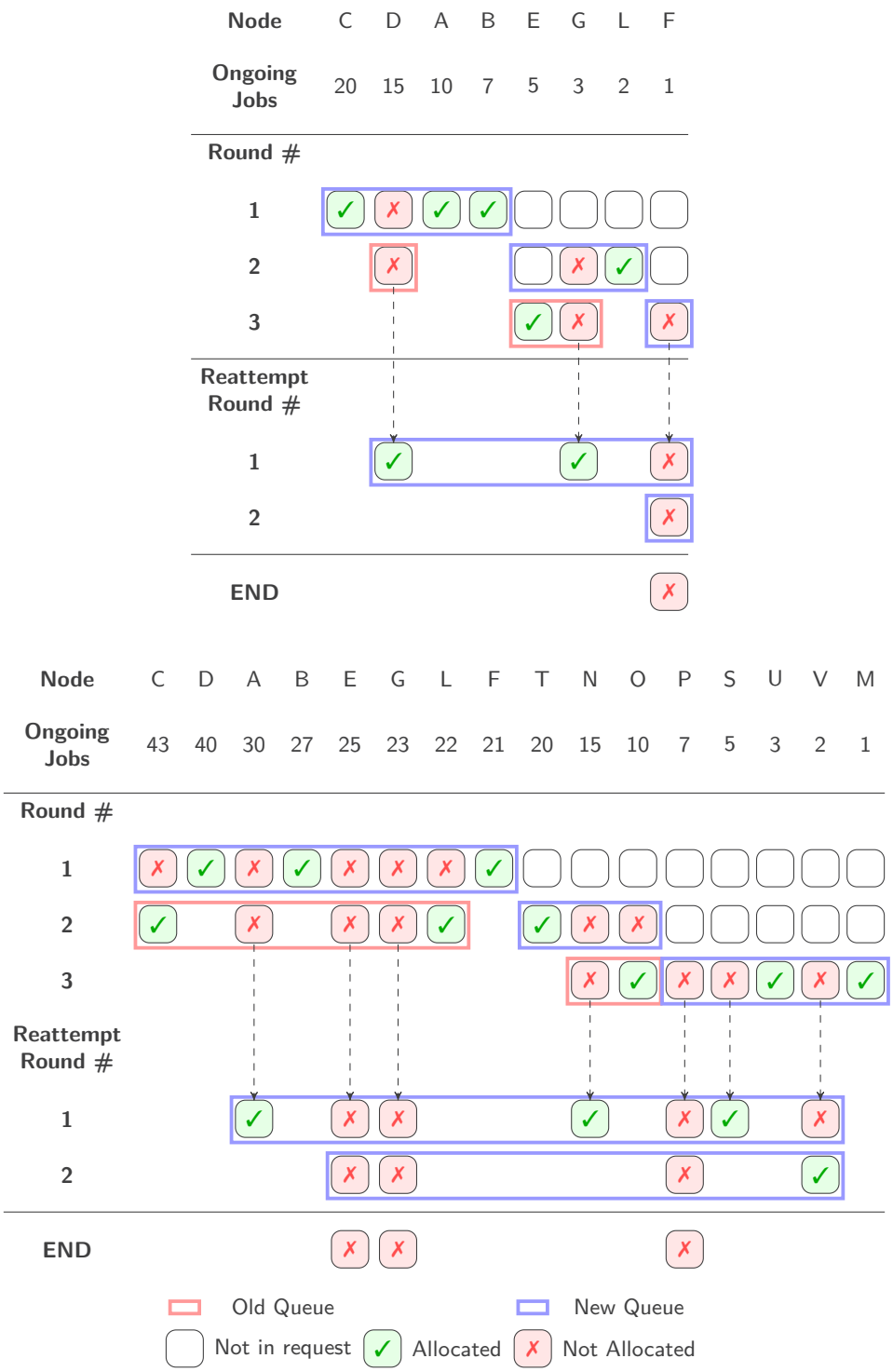


Figure 3.2.: Diagram for building list of nodes to request allocation for two sample scenarios, in which `concurrency` set and number of nodes with having ongoing job steps are respectively 2 and 8 and 4 and 16 for the smaller one at top and larger one at bottom.

Algorithm 1
Algorithm to scrape measurements from remote nodes

```
procedure Scrape(Env, FreqConfig)
    Results ← []
    Timeout ← 0
    for Round from 1 to FreqConfig.ScrapeRounds do
        Timeout ← CurrentUnixTime + Freq.PerScrapeRound
        ProcessData, ChildProcesses, StepdList, ApplicationInUse ← {}
        for all Directories in /proc with Name being a number do
            PID ← Name
            StepdRegex ← [slurmstepd: ([0-9])+.([0-9]+|[a-z]{3,11})]
            ▷ ParseStat reads various status and statistics from /proc/PID and returns the data collected ◁
            ProcessData[PID] ← ParseStat(PID)
            CmdLine ← Read(/proc/PID/cmdline)
            IsMatch, JobId, StepID ← CmdLine matches StepdRegex
            if IsMatch and ValidateStepID((StepID)) is true then
                StepIDVal ← NumericStepID(StepID)
                NCPUAvailable ← ReadCPUSet(PID, Env.CGroupMountPoint)
                StepdList.Append({(JobID, StepIDVal): (PID, NCPUAvailable)})
            else
                ChildProcesses[PPID].Append(PID)
                Command ← Read(/proc/PID/comm)
                ApplicationInUse[PID].Append(Command)
            end if
        end for
        ▷ Establish connection with environment data and set GPU stats for processes in the result. ◁
        FetchGPUStats(ProcessData, StepdList, Env.GPUProvider)
        ▷ Take each PID in StepdList as root and update values to be aggregated data in the subtree. ◁
        Aggregate(StepdList, ChildProcesses, ProcessData, ApplicationInUse)
        Results.Append(StepdList, ProcessData.Filter(PID in StepdList))
    end for
    SendData(Results, Env.WatcherServerInfo)
end procedure
function AllocateScraper(Env, RequestedNodes, FreqConfig)
    AllocReq ← {NodeList: RequestedNodes, TimeLimit: FreqConfig.PerNode}
    AllocReq.UnionBy({MinNodes: 1, MaxNodes: RequestedNodes.Size})
    Wait ← FreqConfig.AllocationMaxWait
    AllocatedNodes ← SLURMAlloc(AllocReq, GetScraperEnv(Env), Wait)
    if Allocated.Size > 0 then
        WaitForRemoteCallOnNodes(Scrape, [FreqConfig], AllocatedNodes)
    end if
    return AllocatedNodes
end function
```

Algorithm 2
Algorithm to select an available account for each partition, if any

```
▷ Returns (Account, Partition) tuple, through essentially simulating SQL
  query, by performing a series of set operations. ◁
function GetNodePartitionAccountMapping(SLURMConfig)
    Require: {Nodes, Associations, Partitions} ∈ SLURMConfig
    AllQos, AllAccounts ← SLURMConfig.Associations.Union(Qos; Account)
    CurUserAssocs ← SLURMConfig.Associations.Filter(User is EffectiveUser)
    UsableQos ← CurUserAssocs.GroupBy(Qos).Union((Account, Qos))
    UsablePartition ← {}
    for all Partition ∈ SLURMConfig.Partitions do
        DeniedQos ← Partition.DeniedQos
        DeniedQos.UnionBy(AllQos.Except(Partition.AllowedQos))
        DeniedAccts ← Partition.DeniedAccounts
        DeniedAccts.UnionBy(AllAccounts.Except(Partition.AllowedAccounts))
        AccountToUse ← NULL
        for all (Account, Qos) ∈ UsableQos do
            if Account ∉ DeniedAccts and Qos ∉ DeniedQos then
                AccountToUse ← Account
                exit for
            end if
        end for
        if AccountToUse ≠ NULL then
            UsablePartition.UnionBy((Partition, AccountToUse))
        end if
    end for
    ▷ The scraper runs for a comparatively short period so prefer fast queues to avoid
      waiting for long jobs to finish. ◁
    UsablePartition.Sort({MaxNodes: LargerFirst, MaxTime: SmallerFirst})
    AssignedNodes ← {}
    NodesInPartition ← {}
    for all (Partition, _) ∈ UsablePartitions in sorted order do
        for all Node ∈ Partition.Nodes.Exclude(AssignedNodes) do
            AssignedNodes.UnionBy(Node)
            NodesInPartition[Partition].UnionBy(Node)
        end for
    end for
    return UsablePartition, NodesInPartition
end function
```

Algorithm 3
Node selection algorithm used by distributor to spawn scrapers across compute nodes

```
▷ Subset of Env will be passed to scrapers to build connections and read configurations. ◁
procedure DISTRIBUTE(RunningJobs, Env, Concurrency, FreqConfig)
    Timeout ← 0
    loop
        WAITUNTILUNIXTIMESTAMP(Timeout)
        SLURMConfig ← GETSLURMCONFIG(Env)
        ▷ Too long to name in single line. ◁
        PAM, NIP ← GETNODEPARTITIONACCOUNTMAPPING(SLURMConfig)
        PartitionAccountMapping, NodesInPartition ← PAM, NIP
        ▷ One entry for each node allocated to a job step. ◁
        JobCntOnNode ← RunningJobs.GroupBy(Node)
        Timeout ← CURRENTUNIXTIME
        for all (Partition, Account) ∈ PartitionAccountMapping in sorted order do
            NodesInPartition[Partition].UseOnly(JobCntOnNode[Node] > 0)
            NodesInPartition[Partition].SortBy({JobCntOnNode[Node]: Larger First})
            FullQueue.Push(NodesInPartition[Partition] in sorted order)
            EffectiveConcurrency ← MIN(⌊Partition.MaxNodes / 2⌋, Concurrency)
            ▷ Accumulate overestimated timing here to avoid stressing allocator when no allocation could be given. Finer estimation could be acheived by accumulating timeout within loops. ◁
            CntNodes ← FullQueue.size
            Dt ← (⌊CntNodes / (2 · EffectiveConcurrency)⌋ + 1) · FreqConfig.PerNode
            Timeout ← Timeout + Dt
            MixedQueue, NewQueue ← {}
            ▷ It would be hard to be allocated to nodes with largest amount of job steps. Try these only once, instead of twice, before adding to mixed queue. ◁
            OldQueue ← FullQueue.PopAtMost(EffectiveConcurrency)
            while FullQueue.NotEmpty() do
                NumNodeMissing ← 2 · EffectiveConcurrency − OldQueue.Size
                NewQueue.Push(FullQueue.PopAtMost(NumNodeMissing))
                ReqNodes ← OldQueue + NewQueue
                Allocated ← SCRAPE(Env, ReqNodes, FreqConfig)
                MixedQueue.Push(OldQueue.PopAll().Exclude(Allocated))
                OldQueue.Push(NewQueue.PopAll().Exclude(Allocated))
            end while
            MixedQueue.Push(NewQueue.PopAll())
            while MixedQueue.NotEmpty() do
                ReqNodes ← FullQueue.PopAtMost(2 · EffectiveConcurrency)
                SCRAPE(Env, ReqNodes, FreqConfig)
            end while
        end for
    end loop
end procedure
```

4. Analyzer: Report Generation

Using raw data collected by the watcher, the analyzer identifies active users during current time period and builds time series for each of their job steps, showing changes of resource usage like GPU utilization, as well as intensity of kernel activities like disk I/O and memory allocation, throughout each sampling timeframe that has sampling intervals kept constant. Queries can then be performed on the series, deriving metric values and flagging problems of submissions with custom policies, in an extensible manner. The results are further processed into both HTML format for individual users to view on-demand through web interface and receive periodically through email, and into JSON format for external applications like pivot table renderer to use. Both summary and on-demand details are included in all user interfaces designed in this project, including email and web versions of reports, as well as multilevel pivot table view of raw data, to communicate issues effectively and suggest most cost-effective changes, in a persuasive way.

Regarding implementation details, an offset is maintained as the starting record ID of the period, and then job steps with record ID larger than the offset are treated as latest ones in the periods, and naturally those users with job steps marked latest are identified as active. In addition, the analyzer maintains an expiry timestamp for the offset, after which the offset will be switched to the ID of last record and therefore starts a new period. To utilize file operation atomicity and help save disk space, the analysis results produced from each analysis run, including HTML and JSON files, are compressed into gzipped tarballs. With this, a script template is provided for users to perform scheduled actions, like sending batch emails, using the latest tarball, with users' scripts interacting directly with individual result files without worrying about details like selecting tarball, extracting, and cleaning up.

4.1. Email Report

In order to active alert users of problems in their jobs, like underutilization of computing resources or that the execution does not seem to be parallelized, which is an important feature in case of being deployed by administrators, the HTML reports are generated under constraints of HTML emails and delivered to users' inbox. This allows users to easily and effectively respond to the report and obtain supports, as replying the email already have the recipient address filled and original report attached, which waives the need of looking for support address and provides context with both summary and processed data, like histograms of CPU or GPU utilization, reducing the time and effort needed for establishing mutual understandings. In addition, this is also helpful when users are using this tool on their own, as it waives the need and burden of setting up

a web server, that requires certain familiarity to deploy and to configure authentication and access control, while clusters usually have existing mail server set up for job status notifications of SLURM to work that can be leveraged by this tool. Regardless, the user can simply download the HTML file and view with local browsers, which still provides aesthetic and navigational features hard to achieve with plaintext files. To make email recipients feel less insecure and to encourage reading the report when it is being sent in batch by administrators, instead of sending as attachment, the report is embedded as body of email with heavy use of anchors to provide a smooth cross-referencing experience in a restricted setting.

As shown in Appendix A, each report generated consists of a Table of Contents showing report structure, with important subsections highlighted, a usage instruction explaining the reason why the user is receiving this email and describing the navigational features of this HTML email, a TL; DR box summarizing problems identified in the report and how they can be fixed, and a section for each analysis. In each of these analysis sections, there is a subsection named metrics explaining columns of result rows and a subsection listing the causes, impacts, and solutions of possible problems identified in the analysis, followed by subsections consisting of result rows, showing detailed data that results in the color-labelled problem list in the last column. In addition to having a subsection showing individual result rows for each new job step in the period, there may also be a subsection aggregating result rows from current and previous periods that have both name and submit line matching one of the new job steps, so to show the characteristics and common problems of different families of jobs the user is recently submitting.

4.2. Types of Analyses

In order to make the measurement records meaningful and useful for end users, the analyzer is built with a set of easily modifiable and extendable analysis information, each containing a database query used to aggregate data and identify problems using this data and given criteria, as well as the description of problems showing user the causes, impacts, and solutions of a problem, which will be described in detail in the following subsections. The analyses in the generated report are ordered following the difficulty of resolving potential issues, with the easiest at top that may simply require adjustments to the amount of resources requested while submitting the job allocation. Prior to executing analysis queries, an in-memory database is created and attached to the database connection for higher performance, and loaded with a table combining records and job allocation information, with a unique value assigned to each time slice for the analysis queries to distinguish among them. By having the preloaded table to be the data source for analyses to use, this reduces redundancy in queries and eases in adding new analyses.

Through resolving the problems identified, the user can gain benefit by reducing the amount of time waiting in job queue and obtain computation result faster, as less computing resources are requested and certain part of job requiring less resources can be executed while the compute-intense part is being queued. Besides, the user can also

obtain performance benefit with alerts of possible inefficiencies in the allocation requests and execution scripts, like inadequate number of CPU cores or level of parallelization. By guiding users to have higher utilization of resources and have less time spent on inefficient work like waiting for disk I/O and dynamic memory allocation, this ensures that higher amount of time is spent on actual computing and hence less time is required to return result, with which related fixed-rate carbon footprints, like those produced by cooling when the job runs for unnecessarily long, can be reduced without affecting users' work, and therefore increases energy efficiency. This ultimately contributes to higher energy efficiency, as the tasks are now completed faster and have less energy consumption from facility, e.g. those from cooling and disk array, associated with them, for the fact that these components have to be always powered but a single task have minor impact on their consumption, while the amount of computation is constant regardless of computing resources available.

4.2.1. Resource Usage

Poor utilization of allocated computing resources can occur independent of whether the job submission uses some script or software written from scratch or scientific software that has been highly optimized and consumes every bit of compute power available to it. For example, the user may forget to specify number of cores available and therefore use a small default value instead, may assume that the software could utilize GPU cards while in fact not, or may simply mistyped `#SBATCH` and `#SBTACH`, leading `sbatch` to ignore this line. Plus, overestimating the amount of RAM required is commonly happening partly due to concerns of OOM kills causing loss of work and waste of time, while it takes extra effort to check the amount of memory actually used. With this reason, this analysis is placed first for its alerts being intuitive while easy and worthwhile to fix, and provides users with observability of the situation by contrasting the amount of computing resources allocated and actually used in a job submission and individual steps in it, in percentile format for easy interpretation and being presented as histogram when the granularity of data allows, as shown in Table 4.1. To aid in quickly locating the problem, nodes that raised certain type of alert is marked with a double asterisk in the respective column. In addition to commonly identified problems like underutilization of CPU cores and GPU cards, this analysis also alerts submissions that appears to have no concurrency by utilizing only one CPU core and no GPU, or those that requested unusually low amount of resources, leading to the fact that their computing capability may not as competent as a regular laptop, whose examples are shown in Table 4.2.

4.2.2. GPU Usage

The demand of high-specification compute cards and GPUs in HPC clusters increased with the raising popularity of machine learning and developments on it required more and faster devices to train high accuracy models. However, the computing power of GPU cards cannot be split into hundreds or thousands of cores as CPU does and therefore has less granularity when sharing, requiring efficient usage and careful selection of card

type to reduce waste of allocated resources. This analysis uses direct or indirect data from GPU driver, such as NVIDIA Management Library (NVML) or those collected by Bright Cluster Manager through this library, to alert users of misuses and suggest them to split job into GPU and pure CPU parts when there is long period of idle observed, for the scarce resource to be better shared among the cluster. Besides the zero-usage alert that flags connection to GPU card driver with no observed utilization, low utilization observed for high percent of time is also used as a criterion for asking users to investigate code submitted for execution or to use cards of lower specification so to ensure jobs are using cards whose specification is matching their need. An example section of GPU is shown in Table 4.3.

4.2.3. System Time Ratio

Since it is unrealistic to write rules to cover all possible misuses and inefficiencies in HPC environment, indicative and suggestive rules like the ratio of time spent in kernel state to that in user time, or system time ratio, is introduced to identify submissions for which profiling and targeted optimization will be helpful, as in the example result shown in Table 4.4. The case study in Appendix B shows a dedicated optimization hinted by system time ratio, using profiling result from `strace`, a tool profiling system call patterns. By helping users to profile their code, administrators can gain understanding on common problems when porting jobs onto the cluster, with which a checklist can be given for users to write code and submission scripts that are efficient for the cluster, ultimately benefiting all users and make cluster run more efficiently.

Job ID	Step	Name	Memory Usage	Timespan	CPU Util	GPU Count	GPU Util	Problems Found
699999		training (1 node)	10.87% (3560 / 32768 MB) source: samples	20.36% of time-limit used actual: 0-04:53:10 available: 1-00:00:00	allocated: 16 cores average: 5 cores actual: 0-20:14:27 available: 3-06:10:40 percentage: 25.89%			**CPU Underusage** **Memory Underusage**
	batch	training /batch (1 node)	10.87% (3560 / 32768 MB) source: samples	[0.00%, 100.00%]	** gpu-2-03 900 samples ncpu inuse percentage 1 57.95% 2 1.14% 3 4.55% 4 2.27% 5 3.41% 6 2.27% 8 3.41% 10 2.27% 11 6.82% 12 10.23% 13 5.68% (16 cores available)	8	** gpu-2-03 90 samples ngpu inuse percentage 0 60.90% 1 39.10%	**GPU Underusage** **CPU Underusage** **Memory Underusage**
700011		experiment1 (1 node)	18.93% (15506 / 81920 MB) source: SLURM	27.63% of time-limit used actual: 1-22:24:40 available: 7-00:00:00	allocated: 40 cores average: 38 cores actual: 73-03:41:55 available: 77-08:26:40 percentage: 94.57%			**Memory Underusage**
	batch	experiment1 /batch (1 node)	18.93% (15506 / 81920 MB) source: SLURM	[0.00%, 100.00%]	** compute-1-01 900 samples ncpu inuse percentage 6 0.23% 7 7.27% 8 37.73% 9 45.80% 10 8.52% 11 0.45% (32 cores available)	0		**CPU Underusage** **Memory Underusage**

Table 4.1.: Anonymized sample underutilization data rows in the resource usage section of the generated report. Word wrapped to fit page size.

Job ID	Step	Name	Memory Usage	Timespan	CPU Util	GPU Count	GPU Util	Problems Found
700009		test (1 node)	82.66% (41133 / 50000 MB) source: SLURM	100.00% of time-limit used actual: 5-00:00:18 available: 5-00:00:00	allocated: 8 cores average: 6 cores actual: 26-06:03:53 available: 40-00:02:24 percentage: 65.53%			**Low Compute Power**
	batch	test /batch (1 node)	82.66% (41133 / 50000 MB) source: SLURM	[0.00%, 100.00%]	compute-2-02 2025 samples ncpu inuse percentage 1 4.29% 2 4.90% 3 11.36% 4 2.93% 5 4.24% 6 2.53% 7 9.44% 8 60.30% (8 cores available)	0		
700055		algo (1 node)	81.92% (163846 / 200000 MB) source: SLURM	60.24% of time-limit used actual: 0-10:50:33 available: 0-18:00:00	allocated: 4 cores average: 1 core actual: 0-10:50:29 available: 1-19:22:12 percentage: 25.00%			**CPU Underusage**
	batch	algo/batch (1 node)	81.92% (163846 / 200000 MB) source: SLURM	[0.00%, 100.00%]	** compute-2-03 135 samples ncpu inuse percentage 1 100.00% (4 cores available)	0		**Low Concurrency** **CPU Underusage**

Table 4.2.: Anonymized sample data rows with low concurrency or low compute power alert, in the resource usage section of the generated report. Word wrapped to fit page size.

Job ID	Step	Name	GPU #	Avg. Util%	No Util	Low Util	Longest Continuous No Util	Avg. SM Clock (MHz)	Avg. Power Usage (Watts)	Problems Found
700028	batch	`training /batch`	`1`	0.00	48	0	0	210.00	26.89	**Completely No Util**
					out of 48 records					
					100.00%	0.00%	0.00%			
700029	batch	`training /batch`	`0`	1.76	96	2	75	384.76	30.88	**Try Splitting**
					out of 103 records					**Investigate GPU Usage**
					93.20%	1.94%	72.82%			

Table 4.3.: Anonymized sample data rows in the GPU usage section of the generated report. Word wrapped to fit page size.

Job ID	Step	Name	(10%, 33%] #	(33%, 66%]	(66%, 100%]	> 100%	Unified Ratio	Problems Found
700033	batch	`collect /batch`	36	881	295	0.00	1471.00	**System Time Ratio**
			out of 1,212 records					
			2.97%	72.69%	24.34%	0.00%	121.37%	

Table 4.4.: Anonymized sample data rows in the system time ratio section of the generated report. Word wrapped to fit page size.

4.2.4. Extensibility

It is clear that being easy to add or modify metrics being recorded and rules for analyses is vital for the tool be applicable in different scenarios and use cases. Therefore, extensibility is taken into consideration when designing the tool and users can add new columns to the record and start collecting and recording these data without concerning too much about implementation details. To do this, they will first append a table modification SQL query to a database migration switch-case, which jumps with regard to current database schema version and performs migration queries for each schema change all the way up to the latest schema version. After this, for the values to be recorded, they will need to add code in scraper to read metric values that are mapped to either subprocesses of `slurmstepd` or job step, and send them back data to server for process, by following existing code as template for appending data as new section in message and inserting data received into the new column just created. No matter whether new columns are added to the database, the users can cleanly extend the analyses by making appropriate changes to the query to add or change alert rules using existing data, or to show more data in generated reports. To do this, they only have to fill out fields of constant structures to inform the analyzer of the change and provide textual content to show in the reports. Both extensions have enough code near sites of changes for reference.

4.3. Web Interface

4.3.1. Backend

The backend is implemented in a relatively simple manner, by having all available results loaded into structures split by period and user, with reports further parsed into different sections and deduplicated to remove redundancy from descriptive texts. To signal creation of and updates to the tarball after finish overwriting it, the analyzer creates empty files with specific filename format indicating a period is updated, which allows filesystem notification set up in the directory to trigger the callback function to reload that period. For handling HTTP requests, it takes authentication header passed by HTTP server and composes JSON response with data selected based on periods in the request and the user's permission.

4.3.2. Frontend

In comparison to HTML emails, in addition to higher consistency across devices, the major advantage of displaying reports with a website is the added interactivity. As shown in Figure 4.1, users can navigate through report sections more easily, not only with the table of contents at left showing current and available sections, but also with the arrow buttons in bottom right and corresponding arrow keys to jump to previous or next section or user, which is especially useful for administrators to scan through the TL; DR section. In addition, the added interactivity also allows the implementation of zooming in individual cells with Ctrl+hover, as in Figure 4.2, to learn more details of a job step without scaling up entire table and requiring horizontal scrolling. Another

feature enabled by this benefit is checking problem explanations and column definitions with popup instead of jumping with anchors, as shown in Figure 4.3. However, it cannot replace emails, as the latter proactively and periodically alert users and keep them aware of the problem instead of requiring them to routinely check the website, while users can also respond easily by clicking on reply button that will also provide support person with context, as aforementioned. Considering that some tips can be helpful for users to improve their productivity, by adopting to shortcut keys and learning command line utilities, as well as to avoid confusions, such as that message `[1]+ Stopped` does not terminate process and that Ctrl+S freezes terminal instead of saving file, a popup named Tips of the Day is included in the interface, as shown in Figure 4.4 with a teaser showing in top right attracting users to click. While tips that better targets individual cluster condition can be identified from user interactions like consulting sessions, initial content are populated with some manually reviewed, edited, and styled ChatGPT 3.5 output for demonstration purpose, with the prompt of "Give me ten less known linux terminal tricks; Summary each of them as informal and short question less than 7 words."

Even given the feature of horizontally navigating through users' reports, it may still be hard for administrators to grasp the cluster's utilization condition quickly, especially when it has to be compared across different periods. Therefore, the analyzer also outputs raw values in JSON format for integrations to use, such as the multi-level pivot table view powered by a JavaScript library named FINOS Perspective[1]. As demonstrated in Figure 4.5 and Figure 4.6, this allows administrators to quickly spot the users constantly underutilizing computing resources, and to compare across time periods to see whether usage condition has improved, for purposes like evaluating the effectiveness of certain policy. This view can also be useful for regular users to see whether certain changes made are effective in resolving certain problem for jobs of specific name, or to simply compare running time of different job submissions, as shown in Figure 4.7.

[1] `https://perspective.finos.org/`

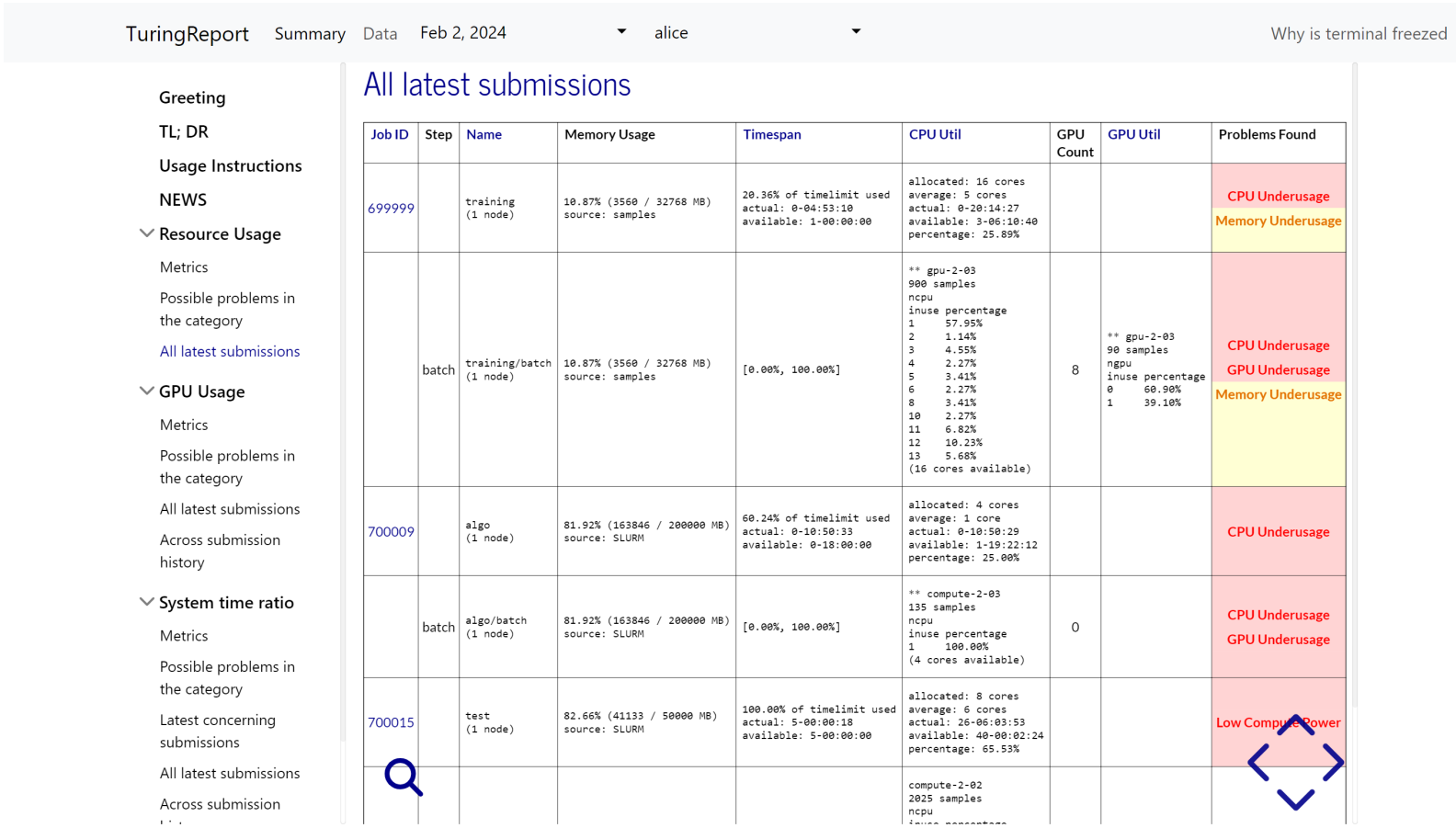


Figure 4.1.: Screenshot of a sample report section viewed through web interface

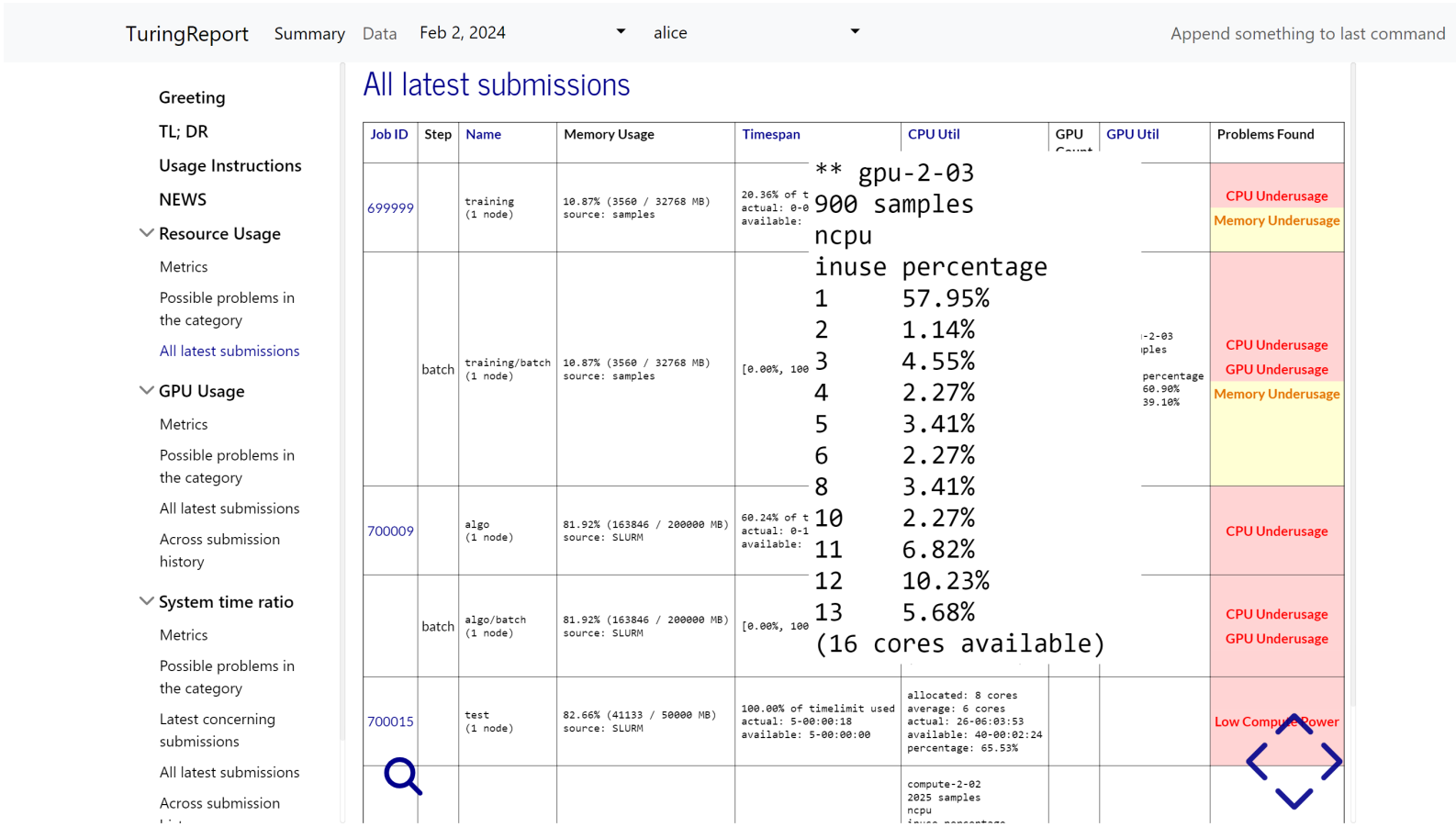


Figure 4.2.: Screenshot demonstrating zooming in individual cell with Ctrl+hover

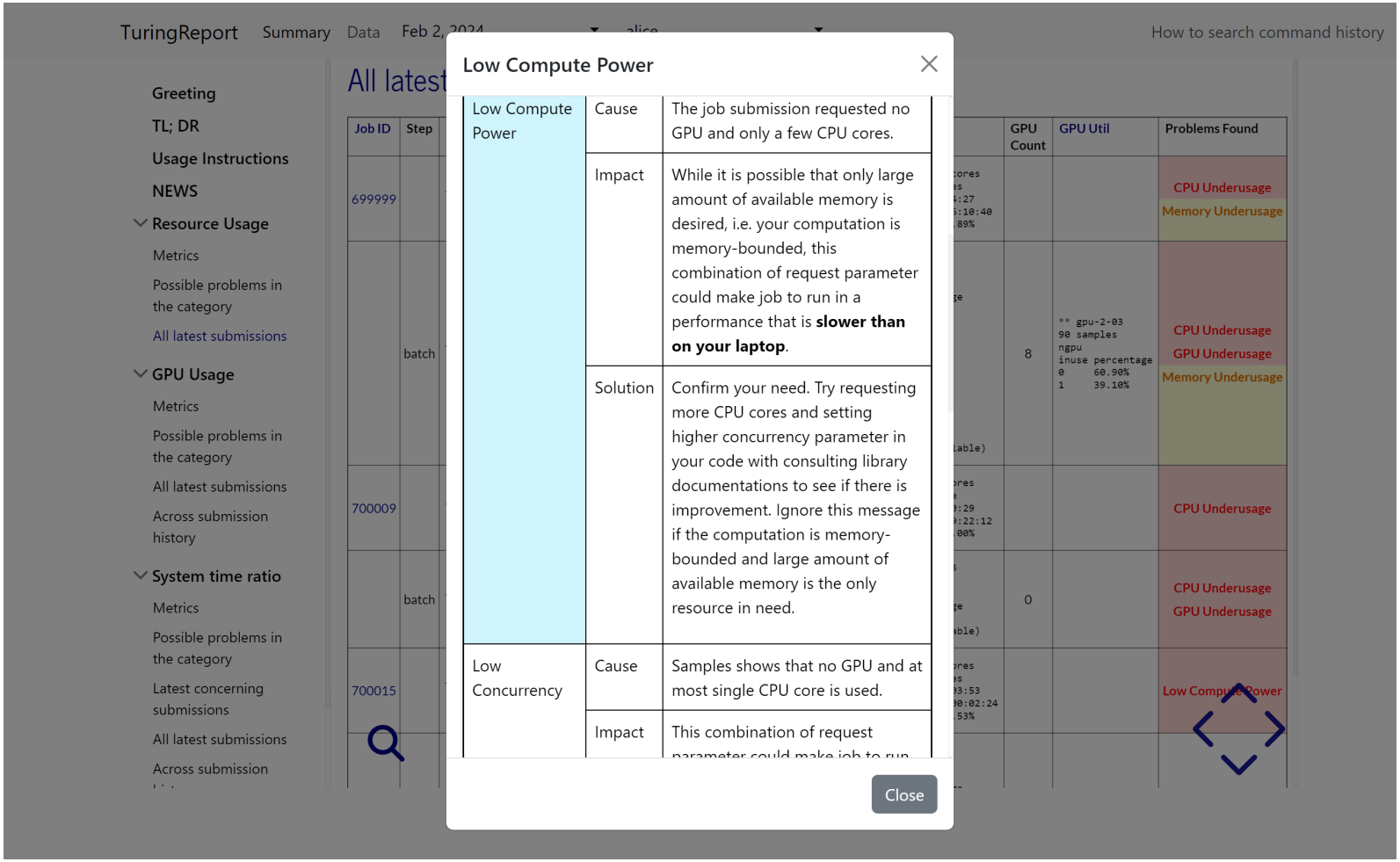


Figure 4.3.: Screenshot showing popups in place of anchors in HTML email

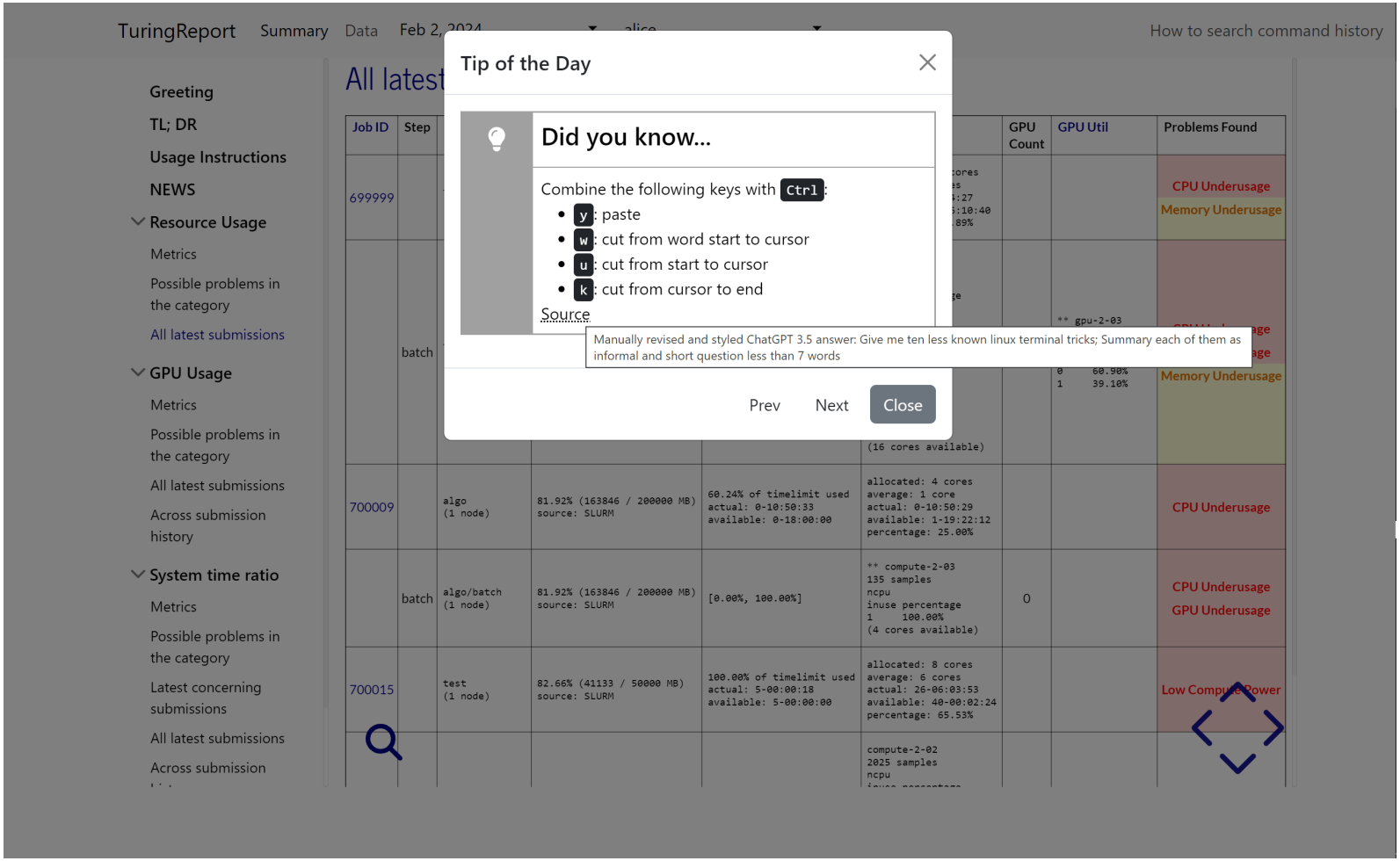
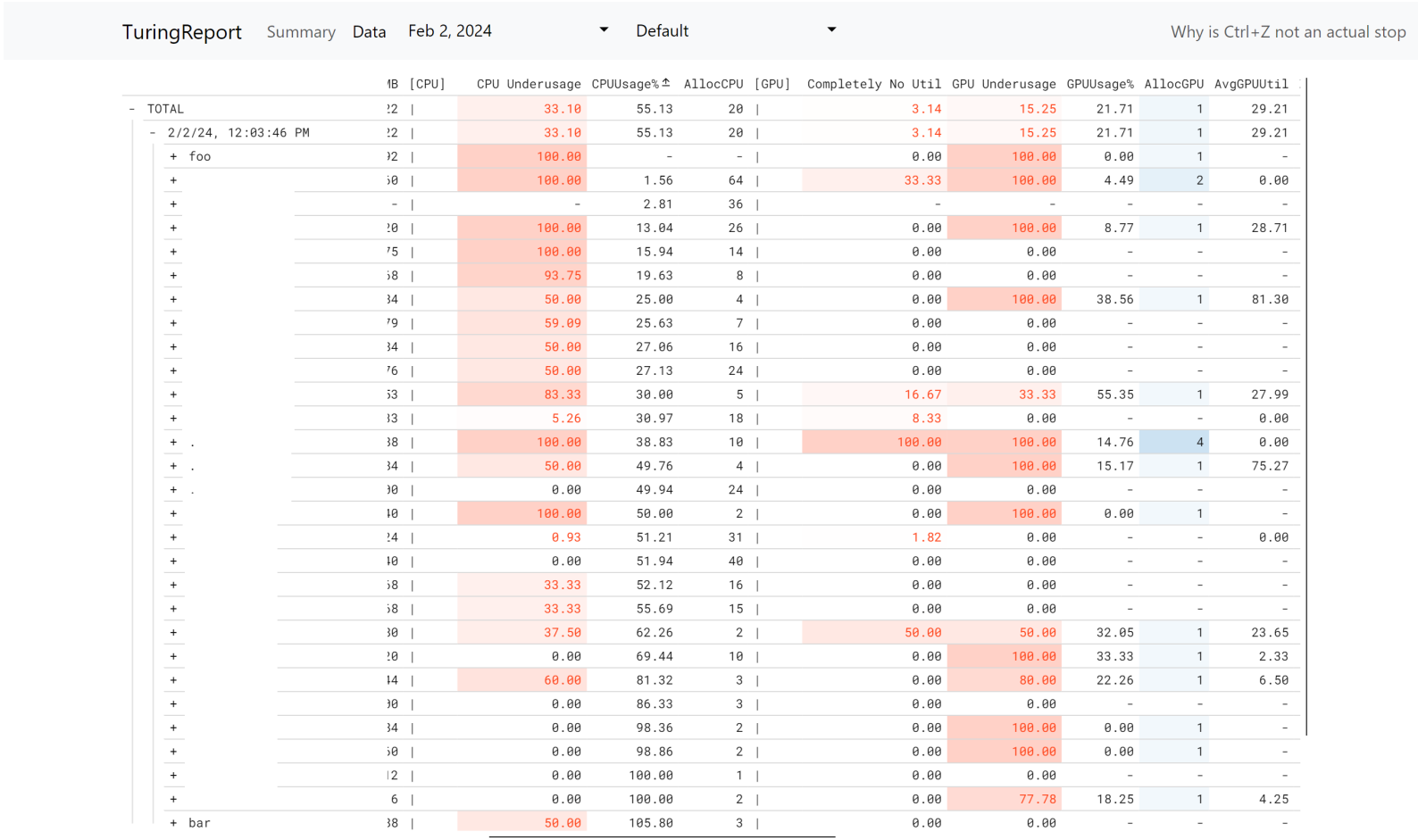
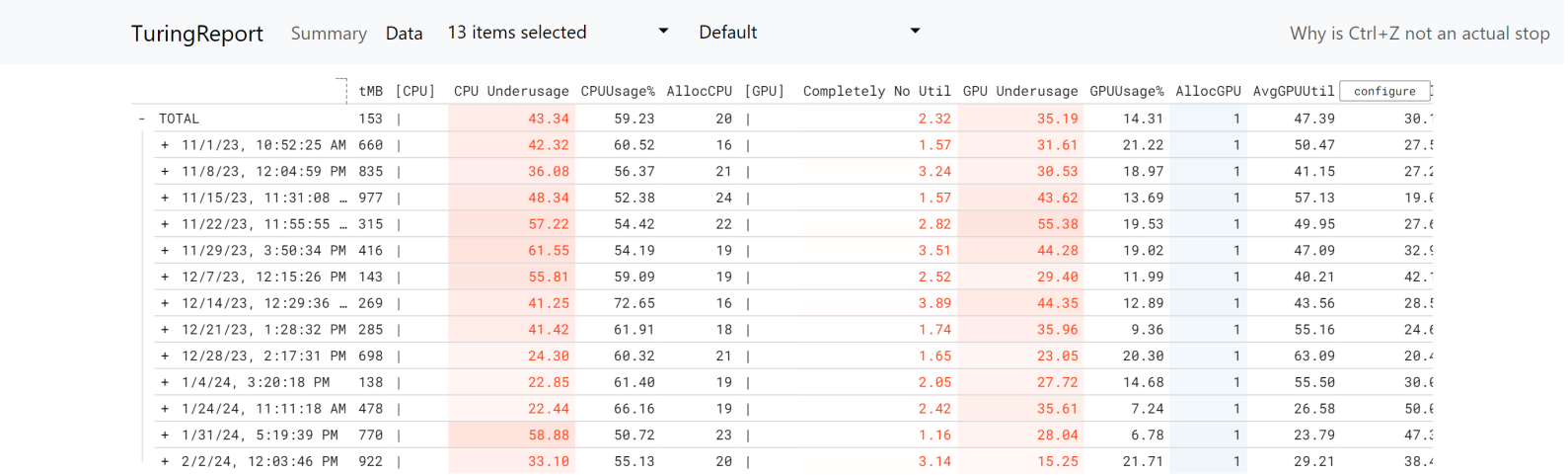


Figure 4.4.: Screenshot demonstrating tip of the day feature



	MB	[CPU]	CPU Underusage	CPUUsage%	AllocCPU	[GPU]	Completely No Util	GPU Underusage	GPUUsage%	AllocGPU	AvgGPUUtil
- TOTAL	12	|	33.10	55.13	20	|	3.14	15.25	21.71	1	29.21
- 2/2/24, 12:03:46 PM	12	|	33.10	55.13	20	|	3.14	15.25	21.71	1	29.21
+ foo	32	|	100.00	-	-	|	0.00	100.00	0.00	1	-
+	30	|	100.00	1.56	64	|	33.33	100.00	4.49	2	0.00
+	-	|	-	2.81	36	|	-	-	-	-	-
+	20	|	100.00	13.04	26	|	0.00	100.00	8.77	1	28.71
+	75	|	100.00	15.94	14	|	0.00	0.00	-	-	-
+	38	|	93.75	19.63	8	|	0.00	0.00	-	-	-
+	34	|	50.00	25.00	4	|	0.00	100.00	38.56	1	81.30
+	79	|	59.09	25.63	7	|	0.00	0.00	-	-	-
+	34	|	50.00	27.06	16	|	0.00	0.00	-	-	-
+	76	|	50.00	27.13	24	|	0.00	0.00	-	-	-
+	33	|	83.33	30.00	5	|	16.67	33.33	55.35	1	27.99
+	33	|	5.26	30.97	18	|	8.33	0.00	-	-	0.00
+ .	38	|	100.00	38.83	10	|	100.00	100.00	14.76	4	0.00
+ .	34	|	50.00	49.76	4	|	0.00	100.00	15.17	1	75.27
+ .	30	|	0.00	49.94	24	|	0.00	0.00	-	-	-
+	10	|	100.00	50.00	2	|	0.00	100.00	0.00	1	-
+	24	|	0.93	51.21	31	|	1.82	0.00	-	-	0.00
+	10	|	0.00	51.94	40	|	0.00	0.00	-	-	-
+	38	|	33.33	52.12	16	|	0.00	0.00	-	-	-
+	38	|	33.33	55.69	15	|	0.00	0.00	-	-	-
+	30	|	37.50	62.26	2	|	50.00	50.00	32.05	1	23.65
+	20	|	0.00	69.44	10	|	0.00	100.00	33.33	1	2.33
+	14	|	60.00	81.32	3	|	0.00	80.00	22.26	1	6.50
+	30	|	0.00	86.33	3	|	0.00	0.00	-	-	-
+	34	|	0.00	98.36	2	|	0.00	100.00	0.00	1	-
+	30	|	0.00	98.86	2	|	0.00	100.00	0.00	1	-
+	12	|	0.00	100.00	1	|	0.00	0.00	-	-	-
+	6	|	0.00	100.00	2	|	0.00	77.78	18.25	1	4.25
+ bar	38	|	50.00	105.80	3	|	0.00	0.00	-	-	-

Figure 4.5.: Screenshot demonstrating pivot table view listing values of different users in the same time period, sorted in ascending average CPU usage, with usernames anonymized to protect privacy. The problem columns are showing percentages of rows having corresponding problem.



	tMB	[CPU]	CPU Underusage	CPUUsage%	AllocCPU	[GPU]	Completely No Util	GPU Underusage	GPUUsage%	AllocGPU	AvgGPUUtil	configure
- TOTAL	153	|	43.34	59.23	20	|	2.32	35.19	14.31	1	47.39	30.
+ 11/1/23, 10:52:25 AM	660	|	42.32	60.52	16	|	1.57	31.61	21.22	1	50.47	27.
+ 11/8/23, 12:04:59 PM	835	|	36.08	56.37	21	|	3.24	30.53	18.97	1	41.15	27.
+ 11/15/23, 11:31:08 …	977	|	48.34	52.38	24	|	1.57	43.62	13.69	1	57.13	19.
+ 11/22/23, 11:55:55 …	315	|	57.22	54.42	22	|	2.82	55.38	19.53	1	49.95	27.
+ 11/29/23, 3:50:34 PM	416	|	61.55	54.19	19	|	3.51	44.28	19.02	1	47.09	32.
+ 12/7/23, 12:15:26 PM	143	|	55.81	59.09	19	|	2.52	29.40	11.99	1	40.21	42.
+ 12/14/23, 12:29:36 …	269	|	41.25	72.65	16	|	3.89	44.35	12.89	1	43.56	28.
+ 12/21/23, 1:28:32 PM	285	|	41.42	61.91	18	|	1.74	35.96	9.36	1	55.16	24.
+ 12/28/23, 2:17:31 PM	698	|	24.30	60.32	21	|	1.65	23.05	20.30	1	63.09	20.
+ 1/4/24, 3:20:18 PM	138	|	22.85	61.40	19	|	2.05	27.72	14.68	1	55.50	30.
+ 1/24/24, 11:11:18 AM	478	|	22.44	66.16	19	|	2.42	35.61	7.24	1	26.58	50.
+ 1/31/24, 5:19:39 PM	770	|	58.88	50.72	23	|	1.16	28.04	6.78	1	23.79	47.
+ 2/2/24, 12:03:46 PM	922	|	33.10	55.13	20	|	3.14	15.25	21.71	1	29.21	38.

Figure 4.6.: Screenshot demonstrating pivot table view comparing average values from multiple periods, with problem columns showing percentages of rows having corresponding problem.

TuringReport Summary Data 13 items selected Default

	NodeCnt	JobLengthHour	TimeLimitHour	Low Concurrency	Low Compute Power
- foobar	1	6.1	24.0	37.09	2.65
- task	1	6.1	24.0	37.09	2.65
+ 11/1/23, 10:52:25 AM	1	9.4	24.0	96.97	0.00
+ 11/8/23, 12:04:59 PM	1	6.6	24.0	100.00	0.00
+ 11/15/23, 11:31:08 AM	1	7.0	24.0	100.00	0.00
+ 11/22/23, 11:55:55 AM	1	4.4	24.0	100.00	0.00
+ 11/29/23, 3:50:34 PM	1	4.5	24.0	100.00	0.00
+ 12/14/23, 12:29:36 PM	1	6.1	24.0	0.00	0.00
+ 12/21/23, 1:28:32 PM	1	5.0	24.0	0.00	0.00
+ 1/4/24, 3:20:18 PM	1	5.9	24.0	0.00	0.00
+ 1/24/24, 11:11:18 AM	1	6.5	24.0	0.00	0.00
+ 1/31/24, 5:19:39 PM	1	4.5	24.0	0.00	0.00
- 2/2/24, 12:03:46 PM	1	5.7	24.0	0.00	12.50
+ 777701	1	7.9	24.0	0.00	0.00
+	1	7.5	24.0	0.00	0.00
+	1	7.5	24.0	0.00	0.00
+	1	7.6	24.0	0.00	0.00
+	1	7.7	24.0	0.00	0.00
+	1	5.9	24.0	0.00	0.00
+	1	6.5	24.0	0.00	0.00
+ .	1	5.5	24.0	0.00	0.00
+ .	1	5.5	24.0	0.00	0.00
+ .	1	4.4	24.0	0.00	0.00
+	1	4.7	24.0	0.00	0.00
+	1	-	24.0	0.00	100.00
+	1	6.7	24.0	0.00	0.00
+	1	6.8	24.0	0.00	0.00
+	1	4.4	24.0	0.00	0.00
+	1	4.2	24.0	0.00	0.00
+	1	5.1	24.0	0.00	0.00
+ 777745	1	5.4	24.0	0.00	0.00

Figure 4.7.: Screenshot demonstrating showing changes in problem distribution in a family of jobs submitted by a user, grouped by time periods, with username, job name, and job numbers anonymized to protect privacy. The values shown in problem columns are percentages of rows having corresponding problem.

Part III.

Conclusions

High performance computing has long been a tool for enabling new breakthroughs in natural science subjects due to its unpaired capability in performing simulation and mathematical operations. The fast-paced development of deep learning, as well as the rising popularity of large language models, all of which require vast amount of computation to achieve state-of-the-art accuracy, further pushed the demand to a new peak. However, our literature review shows that HPC has a significant carbon footprint that will continue to grow in the absence of inventions, raising severe concerns around its sustainability. Despite the amount of energy necessary for the computation to be performed, a significant proportion of energy is not utilized well by cluster users, by underutilizing the resources utilized or running job in a unnecessarily slow manner, both of which hurts energy efficiency. This underutilization is usually not by meant, but due to the fundamental difference between cluster and single machine setting, which the users are typically familiar with, that users do not interactive access to compute nodes in clusters for workload managers isolating them from login nodes to ensure availability of allocated resources, which, in essence, makes the former a black box for cluster users. Unfortunately, most monitoring tools for HPC clusters are not suitable for identifying or addressing those inefficiencies, as they focus on overall operating status rather than individual job characteristics. Moreover, they cannot be easily used by regular cluster users as these tools frequently require deploying database or web servers. To increase energy efficiency – and hence, reduce carbon footprint – without hurting the users' experience and productivity using the cluster, it is vital to guide them to better use the cluster. One way to provide such guidance is to make resource underutilization be observable to them. With this, tools that show resource utilization and automatically identify problems in job submissions grow in demand. In addition, for educating users, administrators also need a tool to know what are the prevailing problems and what changes to ask for users to make.

To keep the usability of both users and administrators under aforementioned considerations, we propose and build a complete data pipeline for this project, encompassing everything from data collection and cluster sampling to reporting via both email and web interface. Our web interface provides a pivot table view for users and administrators to aggregate data easily and see trends and the most prevailing problem in different dimensions. For the users of this new tool to get quickly acquainted, abundant script and `systemd` service file templates are provided with detailed usage instructions. During the 6 months running this tool on our cluster, we have received email inquires indicating the interest from users to improve their job submissions. In addition, patterns can be seen from the pivot table view that some problems identified are resolved starting at certain time point, while administrators now have a better data-driven tool to locate improvement goal. They can also learn about job characteristics easier when communicating with users, without having to examine their submission script first.

Although the tool has already proved useful for both administrators and users to collaboratively make the cluster better utilized, due to limited time and resources available for building the infrastructure to fully support this utility, we enlist a number of points for improvement, that can be taken up by future IQPs:

- Generate suggestive SLURM argument in reports for users to try as a start point.
- Import hierarchical information from SLURM account management interface and allow advisors to see the reports of their students if configured;
- Responding to the cell click event in the pivot table and jump to respective report if the aggregation level is low enough to locate one;
- Immediately sending alert emails to users when serious misuses are observed, such as only one core is being used for a job that has been running for hours;
- Further simplify the process to add analyses, by reading data scraping sequence from a configuration file that instructs where and how values should be fetched;
- Use macro or code generator to automatically populate template for sending and receiving data between scraper and server.

In addition, complementary analysis can be implemented in future works, such as:

- Infer the type of system call that causes an observed increase in system time by checking for concurrent change in values, e.g., total I/O and resident set size;
- Check for frequency of unnecessary operations to free up memory, for example an allocation that happens shortly after freeing memory, of size smaller or equal to freed chunk.

Part IV.

Appendices

A. Sample Report Email

Hi,

Thanks for your attention and hope this letter could help your cluster submissions to be more efficient in terms of both performance and energy! Since the letter serves the purpose of more than communication, but as an observation tool, please do **check out the attached usage instruction for best experience**.

Table of Contents

TL; DR

All **bold** texts in this section are clickable!

- For analysis **Resource Usage**
  - Within 8 records in **all latest submissions**
    - ◇ 1 entry has problem **GPU Underusage** and could be solved by checking your code and allocation request parameters.
    - ◇ 5 entries has problem **CPU Underusage** and could be solved by checking your code and allocation request parameters.
    - ◇ 1 entry has problem **Low Compute Power** and could be solved by checking your code and allocation request parameters.
    - ◇ 1 entry has problem **Low Concurrency** and could be solved by checking your code and allocation request parameters.
    - ◇ 4 entries has problem **Memory Underusage** and could be solved by checking your code and allocation request parameters.

(Continued Figure)

- For analysis **GPU Usage**
  - Within 2 records in **all latest submissions**
    - 1 entry has problem **Completely No Util** and could be solved by checking your code and allocation request parameters.
    - 1 entry has problem **Investigate GPU Usage** and could be solved by requesting a consultation session if needed.
    - 1 entry has problem **Try Splitting** and could be solved by requesting a consultation session if needed.
  - Within 1 record in **across submission history**
    - 1 entry has problem **Investigate GPU Usage** and could be solved by requesting a consultation session if needed.
    - 1 entry has problem **Try Splitting** and could be solved by requesting a consultation session if needed.
- For analysis **System Time Ratio**
  - Within 1 record in **latest concerning submissions**
    - 1 entry has problem **System Time Ratio** and could be solved by could be solved by sending a request to profiling staff to identify bottleneck of the code.

  (Similar summary for analysis across submission history omitted)

Check out the instructions below for the best way of reading this summary letter.

Usage Instructions

- For best experience, use browser-based email rendering engine. Certain email clients like Outlook Desktop have the functionality of opening the email in browser.
- This letter is designed for you to have a better observability of the jobs and identify problems related to resource usage, including underusage or misusage, so that you could **use less** to have the same effect or spot possible **bottlenecks** of the submissions. This not only helps you to better utilize the resources and potentially speed up your jobs, but also contributes to a collective effort of taking up what really need and make queues move faster. All these could contribute to a greener computing.
- Internal links are heavily used in this letter and it is **strongly** advised to use them to avoid scrolling overwhelmingly. **All** text in table of contents and headers are clickable and would respectively bring you to corresponding section or back to table of contents. When being lost in result rows, click on the values of **Job ID** and you would be sent back to section header. Besides, all of the **identified problems** and many of the **column headers** are also **clickable**, which could bring you to a detailed explanation of these names. Please submit a feedback if anything you don't understand is not explained well, including the usage of this letter.

(Continued Figure)

- The highlighted and bolded sections in table of contents are suggested to read first for getting the most out of this letter. Following many users' convention, the measurements are also aggregated by job name across the entire measurement dataset collected so far for a comprehensive overview of job characteristic. Besides identifying problems, the raw data contributing to identifying the problems are also included for your reference.
- Make sure to complete the feedback form for this summary letter to be continuously improved and bring you more valuable information!

NEWS

We have added a series of new GPU nodes last week!

Resource Usage

This analysis identifies resource allocation misuses for you to set allocation request parameters that better fits the actual need and benefit the submission by having requests allocated faster or lowering the risk of having the jobs killed by using more resources than allocated.

Metrics

Job ID	For results aggregated across history runs, the Job ID and Step fields shows a representative job step that has the same job name and submit line, which could be fetched by running `sacct -j . -o Name,SubmitLine`
Name	This field is formed by joining job name and step name with slash.
Timespan	The timing of result row relative to parent constraint. It compares allocated time and actual consumption for jobs, and shows the timing of steps relative to the time window of their parent jobs. For example, a result of [25%, 75%] for a step shows that the step started after a quarter of the job's actual time consumption has elapsed, while ends after running for as long as half of the job's actual time consumption.
CPU Util	The amount of CPU cores used is an average calculated from aggregated CPU time and actual time consumption and does not show peak usage.
GPU Util	The GPU usage information is derived from sampled data and may not represent the full picture.

(Continued Figure)

Possible problems in the category

Low Compute Power	**Cause**	The job submission requested no GPU and only a few CPU cores.
	Impact	While it is possible that only large amount of available memory is desired, i.e. your computation is memory-bounded, this combination of request parameter could make job to run in a performance that is slower than on your laptop.
	Solution	Confirm your need. Try requesting more CPU cores and setting higher concurrency parameter in your code with consulting library documentations to see if there is improvement. Ignore this message if the computation is memory-bounded and large amount of available memory is the only resource in need.
GPU Underusage	**Cause**	Resources allocated is not fully used.
	Impact	Generally, this increases the difficulty for the request to be satisfied, while also keeps unused resources unavailable to other jobs for the length of job and therefore lengthen the queue. Remember sometimes your other jobs could also be in the queue waiting for resources! The combined result of zero GPU utilization and low amount of average CPU cores would bring the submission to be running at extremely poor performance.
	Solution	Adjust allocation request with referring to the usage info provided.
CPU Underusage	Refer to GPU Underusage section above.	
Memory Underusage	Refer to GPU Underusage section above.	

All latest submissions

See Tables 4.1 and 4.2 for examples.

GPU Usage

This analysis helps in making submissions' **GPU usage condition more observable** and provides suggestions on requesting GPU resource so to shorten allocation turnaround time, make allocations better utilized, and save energy.

Metrics

Job ID	For results aggregated across history runs, the Job ID and Step fields shows a representative job step that has the same job name and submit line, which could be fetched by running `sacct -j . -o Name,SubmitLine`
Name	This field is formed by joining job name and step name with slash.
GPU	Index of GPU on machine.
Average Util%	Consider lowering GPU constraint or using lower GPU specification when in low utilization for easier allocation and energy saving, or specifying better GPU in case of high average utilization to get job done faster.
Low Util	Low is currently defined as utilization values in the range of (0, 12.5%].
Longest Continuous No Util	Number of continuous measurements showing zero utilization of the GPU streaming multiprocessor.
Average SM Clock (MHz)	Average streaming multiprocessor clock provided for reference.
Average Power Usage (Watts)	Average power usage provided for understanding environmental impact. As a reference, GPUs typically use 10 Watts when in idle. This does not take external power consumption into calculation, like those for cooling.

(Continued Figure)

Possible problems in the category

Completely No Util	**Cause**	The submission is **never** observed to be utilizing GPU.
	Impact	This causes unnecessary energy consumption by waking GPU from idle mode. In addition, it possibly performed heavy computing with less CPU, which would take even **longer** to complete than a pure CPU submission. It would also block other jobs from using the device for equally long time and stress the queue.
	Solution	Check for code and documentation to ensure the computation is using GPU.
Try Splitting	**Cause**	This submission is having a high percentage of longest zero utilization, indicating that there possibly exists segment of code that is **running for long time while not utilizing any GPU resource**.
	Impact	Allocations without GPU is generally faster to be allocated so that the preparation work for computing with GPU could be performed while waiting for allocation during peak time of GPU usage. This could also allow wasted GPU cycles to be used on other jobs and **utilize energy better**, while also helps the queue to progress faster and shorten the turnaround time waiting for a GPU. Remember your task could be the one waiting for GPU next time so let's be involved in this optimization!
	Solution	Identify code segments running long while utilizing no GPU and split execution into tasks requesting GPU and no GPU. The task depedency could be set while submitting jobs with argument `--dependency=afterok:(jobid)`. Use accurate time limit for smooth transition from one job to another.

(Continued Figure)

Investigate GPU Usage	**Cause**	This submission is observed to be utilizing GPU but in a **low utilization for long period**. This possibly indicates **inefficacies in GPU usage**, like bottlenecks in the pipeline moving data to GPU, or the computation is light enough that GPUs of lower specification or parallized pure CPU computation would fulfill the need while being easier to be allocated and uses less energy.
	Impact	Checking for bottlenecks could help the submission to complete in a faster and more efficient manner. Choosing appropriate resource combination reliefs unnecessary constraints, so that the allocation could be assigned quicker, while avoid blocking jobs with real demands of high specification hardware.
	Solution	Compare time consumption on each part of computation, like that of loading data from disk and preprocessing, moving data to GPU memory, and computing with GPU. Check for bottlenecks in the pipeline. Try GPU of lower specifiction if GPU type is specified in the allocation request.

All latest submissions

See Table 4.3 for an example.

Across submission history

Similar to Table 4.3, with results rows of same name and submit line, both from the latest submissions and previous data, being aggregated together.

System Time Ratio

This analysis identifies job submissions that are likely to be less computationally effective but may not have been captured by other analyses, possibly due to problems that are not previously identified and therefore have no specific rules set up, or those that could not be determined with available metrics.

(Continued Figure)

Metrics

The columns are the number of sampled time slice having ratios in the shown range when the time spent on system requests (system time) is divided by the time spent on user computations (user time). These ratios show how significant the system time is when compared to user time.

Job ID	For results aggregated across history runs, the Job ID and Step fields shows a representative job step that has the same job name and submit line, which could be fetched by running `sacct -j . -o Name,SubmitLine`
Name	This field is formed by joining job name and step name with slash.
>100%	The value shown in this field is the sum of all ratios greater than 100%.
Unified Ratio	The value shown in this field shows the **underestimated** equivalent of the system-time to user-time ratio based on significant ratios shown in previous columns. A unified ratio that equals to the number of measurements as shown in denominator equivalents to a 1/3 system-time to user-time ratio and scales linearly.

Possible problems in the category

System Time Ratio	**Cause**	The process is usually waiting for certain resource to be ready or certain device operation to be done while being counted toward system time. High ratio may indicate a bottleneck of the submitted job.
	Impact	Given the nature of system time, it is **less helpful for progressing the actual computation** and should be reduced to improve the overall computing efficiency.
	Solution	Since system time is highly coupled with the architecture behind, it is suggested to **submit your job for profiling**. This should be **low effort** and would be beneficial to both you and other cluster users.

Latest concerning submissions

See Table 4.4 for an example.

Across submission history

Similar to Table 4.4, with results rows of same name and submit line, both from the latest submissions and previous data, being aggregated together.

You are receiving this email because you have recently submitted jobs to the cluster. For any questions, concerns, or to unsubscribe, please contact administrator. Top

B. Case Study: Optimization Hinted by System Time Ratio

The sample program being profiled uses Tensorflow to perform link prediction, by utilizing methods like `FullBatchLinkGenerator` and `GCN` (Graph Convultion Network), with initial test showing that system time is 10.74s and user time is 12.19s when the number of epoches is set to be 50, while system time become 17.35s and user time become 16.38s when that is set to be 100, and system call `munmap` took 38.51% and 51.21% of time when being profiled by `strace`. Considering the significant performance penalty of system call for switching into kernel state and possible lockings involved in the process, the project started out by profiling the system call pattern of the sample code. The result indicates that each `munmap` call is taking about 1ms to perform in average and the amount of requests is linear to epoch amount, causing significant slowdown of the program. With the further finding that most of these calls are made by functions including `malloc`, `memalign`, and `posix_memalign` in `glibc` (referred as `malloc` below) to allocate chunked memory, a recycle and reuse mechanism for these chunks is designed as detailed below with utilizing the unused spaces within the chunk with pleasing performance and optimization result.

B.1. Overlaid Memory Management

Since, as profiling the system calls from running a sample Python script turns out, `munmap` is the system call running the slowest while does not serve as a synchronization primitive, individual calls to it, as well as its sibling call `mmap`, are timed and plotted as in Figure B.1. It could be identified from the figure that the first half of calls are of relatively light colors, indicating that they runs relatively fast, with the difference of having used `MAP_NORESERVE` flag in the `mmap` call that specifies no swap space should be reserved and risk running out of physical memory. However, in contrast, the latter half working on fitting the model is using `malloc` to repeatedly allocate and free same set of memory in each epoch with large size parameter and are running much slower, as indicated by the deep color. Since these memory regions are backed by `mmap` and managed by `malloc`, a design based on the chunk structure `malloc` uses is proposed here to relief the problem.

B.1.1. Memory Recycling and Reusing

Given the fact that the same set of memory chunks are to be requested in the near future after being freed, while the free operation comes with significant overhead coming from

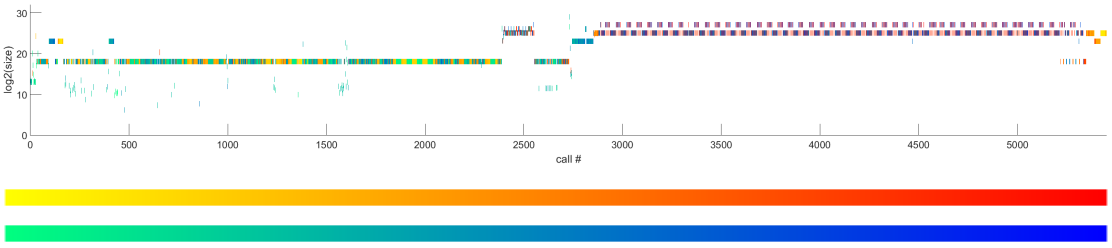


Figure B.1.: `mmap` and `munmap` pattern with regard to request size. Color indicates time taken to perform the system call and the legends shows color of fastest call at leftmost side and slowest at rightmost side. The yellow-red colormap represents the timing of `munmap` operation while the green-blue one represents that of `mmap` operation.

switching context into kernel state, acquiring locks in library and operating system, as well as causing page faults, it would be better to manage these chunks in user space. This led into the solution of trying to recycle freed chunk before calling `free` in `glibc` and trying to reuse recycled chunk of the same specification, namely size and alignment, before calling the actual implementation in `glibc`. This is achieved by making our library a preload library so that the runtime linker prioritizes binding symbols onto our functions.

However, as shown in Figure B.2, the `size` field is actually storing value that computes into the size of entire chunk instead of requested size, making it hard to limit their reuse to requests with exactly the same specification without storing extra information. Fortunately, with the source code of `glibc` indicating that *Arena* and *PrevInUse* bits are always ignored when *MMapChunk* bit is set, it is guaranteed that there is always a space outside header region and user memory that is capable of storing value that could compute into the requested size. Within the chunk from `mmap`, the regions before header consisting of `prev_size` and `size` fields and after user region *possibly* exists but are used by neither the library nor the user, as demonstrated in Figure B.2. Since these free space come from the alignment need of `glibc` and the need of rounding up to page size for being a vaild `mmap` call, they may not be large enough to store a `size_t`-typed value or even do not exist all. However, the presence of both of these extreme cases also indicate there are really small difference between the size of entire chunk and requested size (less than 8 bytes to no difference), and hence the data could be stored in 1 byte as a difference or 1 bit as a flag, as in C1, C2 and N fields in Figure B.3. One exception here is aligned allocation like `memalign` or `posix_memalign`, in which `glibc` automatically adds `alignment` bytes to the requested size to ensure there is an aligned address to return at the worst case, in which case the alignment value could be stored along with the size requested by the user, achieving the goal of storing exact specification of allocation.

For the selection of data structure to maintain the records, an implementation that does efficient exact matching for recently recycled chunks while automatically discard

long unused chunks would be desirable, which exactly resembles an LRU buffer. While linked list has been considered as one option, their cursors in many of the concurrent implementations are subject to invalidation when deletion happens, not to mention keeping the query under appropriate amount of time with lengthening chains and being not cache-friendly are also problematic for applying to this purpose. After evaluating the characteristics of the possible choices, to ensure efficiency, the recycle and reuse processes are implemented by maintaining a pool of recycled chunks using atomic primitives under relaxed memory order. By having the bitwidth of cursor to match that of pool size, it enables unsigned cursor to automatically wrap to zero with natural overflow when incrementing out of bound. Meanwhile, the downside of this implementation is that it requires the pool size to be chosen from a few fixed powers of 2 to match the bitwidth. The size of 256 is selected here since the next value 65,536 would be too large to traverse and confirm there is no matching chunk. For the limited pool size, only large enough chunk is recycled so that the cache would not be invalidated so fast, while also ensuring that the time saved from calling less `munmap` exceeds the overhead from this implementation. The correctness of this implementation is guarded by implanting data onto allocated memory region that does not intersect with where `glibc` and user program would access.

B.1.2. Results

The results from this optimization comes out pleasant, mainly because it optimized out one heavy factor that led the system time to be almost linear to number of epoches training a model. For the distribution of epoches used to sample performance data, the X-axises of the figures are set to be logarithmic of number of epoches to make them spread evenly on the axises, and therefore what seemed to be growing exponentially in the figure is actually linear in the original scale.

From the perspective of time consumption, the implementation turns the time spent on the `munmap` call from being linear to the number of epoches to be staying in a relatively constant level, and in the meantime also avoided the call from taking increasingly higher proportion of overall system call time, as shown in Figure B.4. For the overall time, the use of this implementation resulted in much lower system time with only slight increase in user time, as in Figure B.5, whose combined effect is nearly halving the total execution time as in Figure B.6 for larger epoches, although having slightly slowed down the execution for runs of less than 20 epoches, as detailed in Figure B.7.

As for memory consumption, as shown in Figure B.8, this implementation used about 200∼300M more of memory than the original version at peak, possibly due to the chunks in recycle pool but is tolerable. However, the effect of this implementation on reducing page fault and saving time is significant, especially as the number of epoches goes up.

B.2. Other Possible Slowdowns

1. After resolving the slowdown from `mmap`, `unlink` becomes the slowest system call, with time impact much more trivial than `munmap` although still being linear to

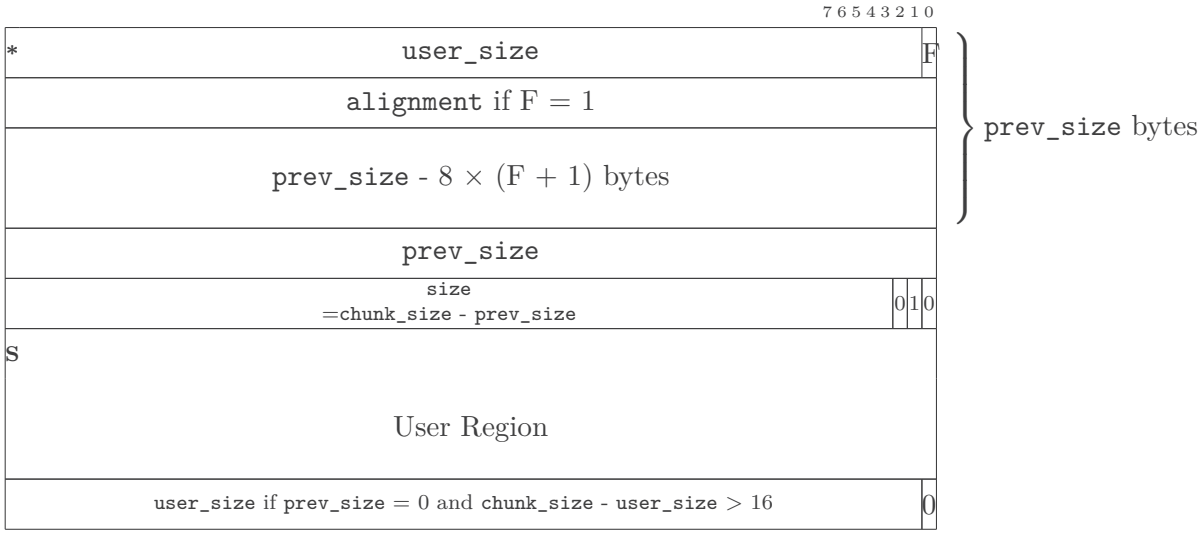


Figure B.2.: Noncompress version of implanting, where the mmaped region starts at the byte as indicated by * and the returned address points to the byte as indicated by S. It discards the highest bit with the assumption that this implementation is not handling such a large size, and implant at either start of chunk or immediately after the user region, with the former one preferred to avoid extra page fault when the last page is never accessed by the user program. It also implants an alignment value if there is space available and set F to 1 in this case. Compressed version as in Figure B.3 shall be used when neither of the spaces are available.

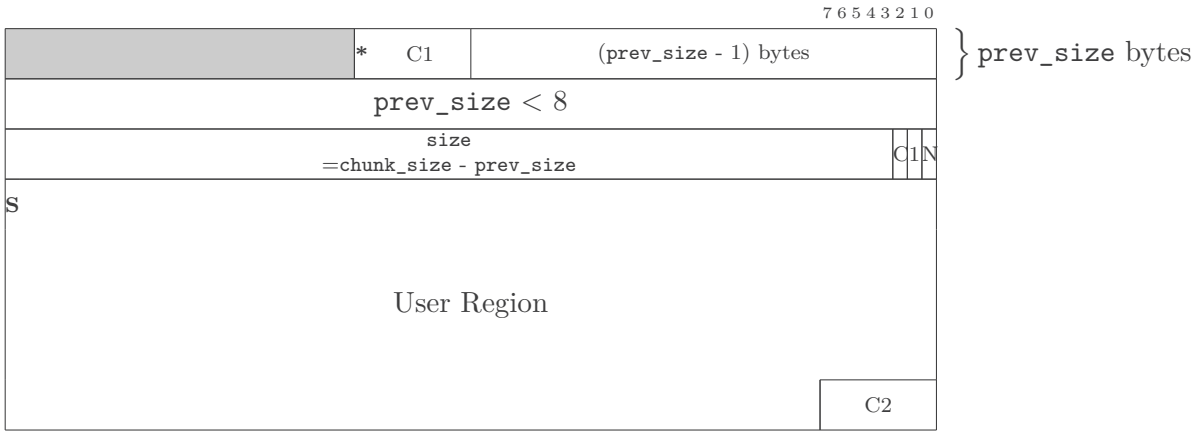


Figure B.3.: Compressed version of implanting, where the mmaped region starts at the byte as indicated by * and the returned address points to the byte as indicated by S. C1 as indicated here is the first byte of mmaped region and C2 as indicated here is the byte immediately following the end of user region. It implants the difference between available size and requested size, which is guaranteed to be less than 8, at either C1 or C2 with C1 preferred and set C=1 and N=0. By preferring C1, it avoids an extra page fault when user program never accesses the last page. If neither space is available then C=0 and N=1 are set, indicating that the size available is exactly the same as size requested.

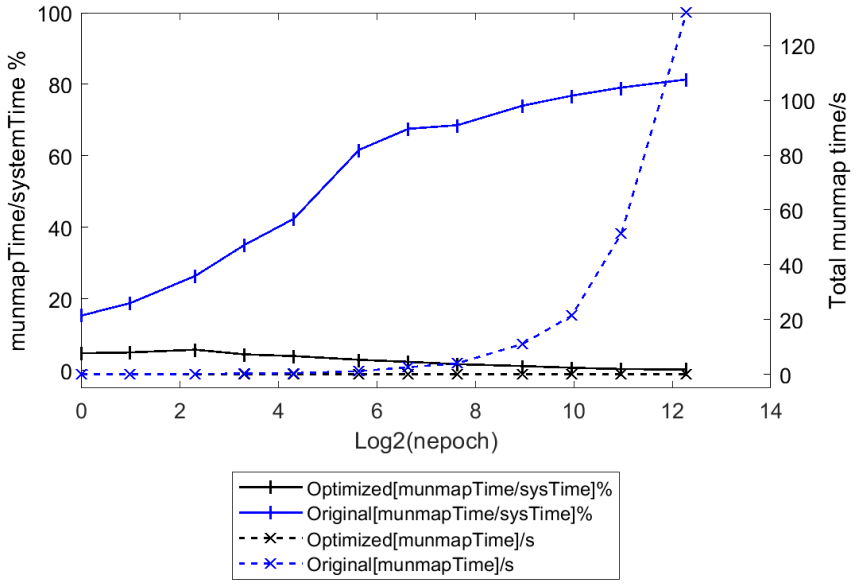


Figure B.4.: Time spent on `munmap` in original (blue) and optimized (black) versions, as well as their percentage in overall system call time (dashed), for different amount of epoches in exponential spread

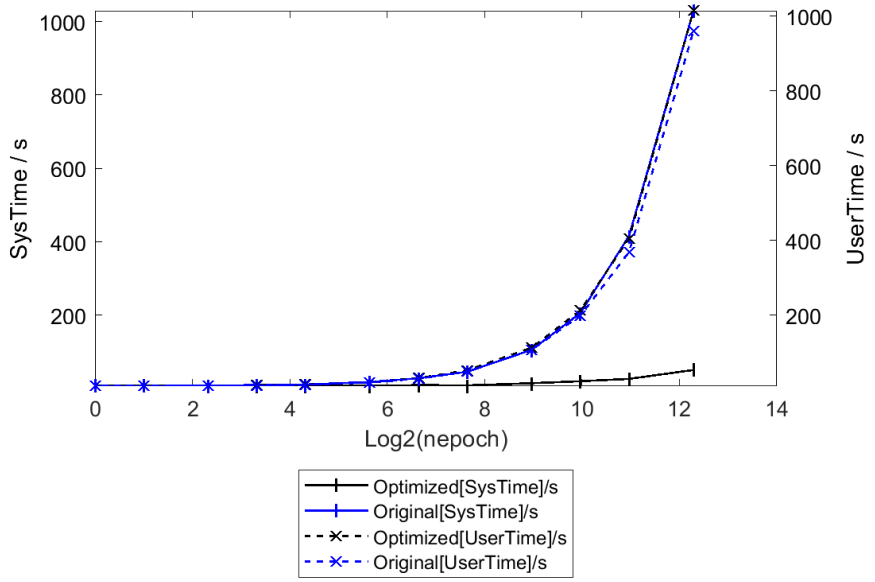


Figure B.5.: User time (dashed) and system time in original (blue) and optimized (black) versions for different amount of epoches in exponential spread

amount of epoches. This is probably caused by Python making copies of scripts to `/tmp` directory and simply deleted them after done using, instead of reusing the filenames from `mktemp`. This could possibly be fixed by handling `mktemp` and `unlink` to recognize and reuse discarded filenames.

2. Checkpointing, which stores current program state into a file, could also been one reason for slowing down the overall performance for its blocking nature. However, there is currently no sample code handy that is reproducing this situation.

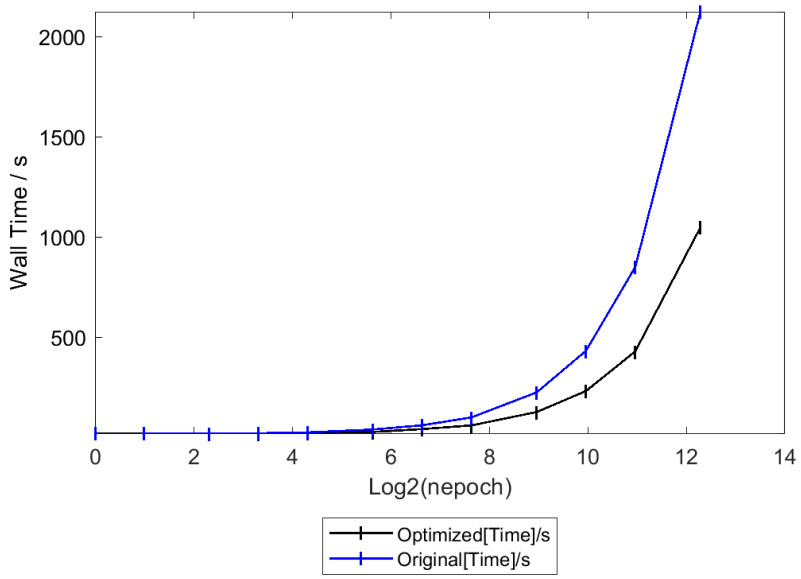


Figure B.6.: Wall clock time in original (blue) and optimized (black) versions for different amount of epoches in exponential spread

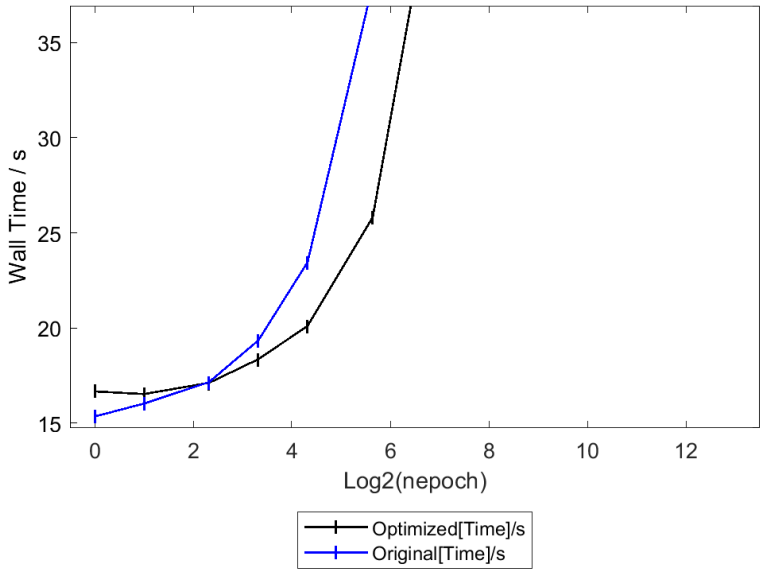


Figure B.7.: Wall clock time in original (blue) and optimized (black) versions for smaller epoches

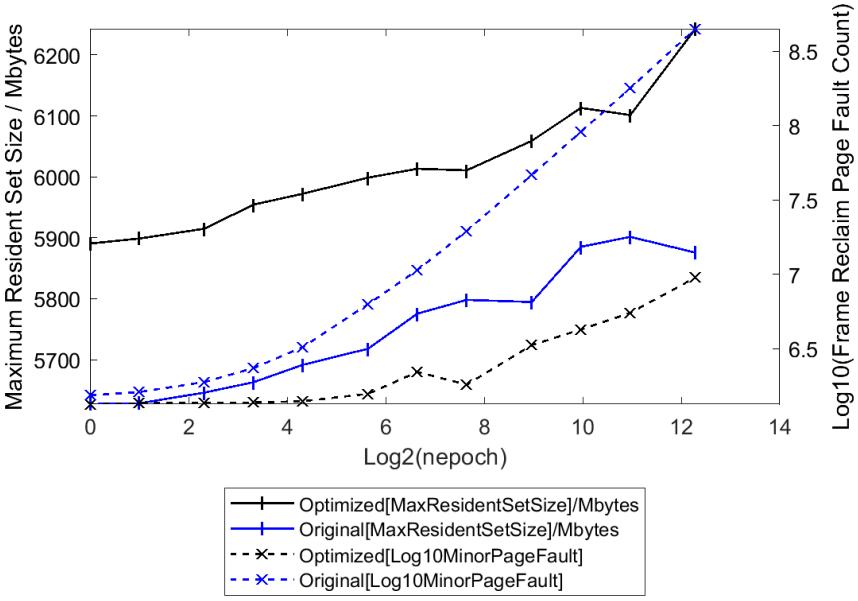


Figure B.8.: Frame reclaim page fault count (dashed) and memory usage as represented by maximum resident set size in original (blue) and optimized (black) versions, for different amount of epoches in exponential spread

References

[1] William (Bill) Allcock et al. "Challenges of HPC monitoring". In: *State of the Practice Reports*. SC '11. New York, NY, USA: Association for Computing Machinery, Nov. 2011, pp. 1–6. ISBN: 9781450311397. DOI: 10.1145/2063348.2063378. URL: https://doi.org/10.1145/2063348.2063378 (visited on 10/09/2023).

[2] Ankit et al. "Electronic waste and their leachates impact on human health and environment: Global ecological threat and management". In: *Environmental Technology & Innovation* 24 (Nov. 2021), p. 102049. ISSN: 2352-1864. DOI: 10.1016/j.eti.2021.102049. URL: https://www.sciencedirect.com/science/article/pii/S2352186421006970 (visited on 12/11/2023).

[3] Pierre Augier et al. "Reducing the ecological impact of computing through education and Python compilers". en. In: *Nature Astronomy* 5.4 (Apr. 2021), pp. 334–335. ISSN: 2397-3366. DOI: 10.1038/s41550-021-01342-y. URL: https://www.nature.com/articles/s41550-021-01342-y (visited on 10/09/2023).

[4] Shajulin Benedict. "Energy-aware performance analysis methodologies for HPC architectures—An exploratory study". In: *Journal of Network and Computer Applications* 35.6 (Nov. 2012), pp. 1709–1719. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2012.08.003. URL: https://www.sciencedirect.com/science/article/pii/S1084804512001798 (visited on 10/09/2023).

[5] Shajulin Benedict. "Performance issues and performance analysis tools for HPC cloud applications: a survey". en. In: *Computing* 95.2 (Feb. 2013), pp. 89–108. ISSN: 1436-5057. DOI: 10.1007/s00607-012-0213-0. URL: https://doi.org/10.1007/s00607-012-0213-0 (visited on 10/09/2023).

[6] Francesco Beneventi et al. "Continuous learning of HPC infrastructure models using big data analytics and in-memory processing tools". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. ISSN: 1558-1101. Mar. 2017, pp. 1038–1043. DOI: 10.23919/DATE.2017.7927143. URL: https://ieeexplore.ieee.org/abstract/document/7927143 (visited on 10/10/2023).

[7] Raúl Cassia et al. "Climate Change and the Impact of Greenhouse Gasses: CO2 and NO, Friends and Foes of Plant Oxidative Stress". In: *Frontiers in Plant Science* 9 (Mar. 2018), p. 273. ISSN: 1664-462X. DOI: 10.3389/fpls.2018.00273. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5837998/ (visited on 12/11/2023).

[8] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. "Data Center Energy Consumption Modeling: A Survey". In: *IEEE Communications Surveys & Tutorials* 18.1 (2016), pp. 732–794. ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2481183. URL: https://ieeexplore.ieee.org/document/7279063/ (visited on 10/08/2023).

[9] Sonja Filiposka, Anastas Mishev, and Carlos Juiz. "Opportunities and Challenges for Green HPC". en. In: *ICT Innovations 2014*. Ed. by Ana Madevska Bogdanova and Dejan Gjorgjevikj. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2015, pp. 45–54. ISBN: 9783319098791. DOI: 10.1007/978-3-319-09879-1_5.

[10] Charlotte Freitag et al. "The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations". In: *Patterns* 2.9 (Sept. 2021), p. 100340. ISSN: 2666-3899. DOI: 10.1016/j.patter.2021.100340. URL: https://www.sciencedirect.com/science/article/pii/S2666389921001884 (visited on 12/11/2023).

[11] Pablo García-Risueño and Pablo E. Ibáñez. *A review of High Performance Computing foundations for scientists*. en. May 2012. DOI: 10.1142/S0129183112300011. URL: https://arxiv.org/abs/1205.5177v1 (visited on 12/03/2023).

[12] David Guyon et al. "How Much Energy Can Green HPC Cloud Users Save?" In: *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. ISSN: 2377-5750. Mar. 2017, pp. 416–420. DOI: 10.1109/PDP.2017.62. URL: https://ieeexplore.ieee.org/document/7912681/;jsessionid=tgctRYP8C7pg5xecbxi4jRDHW4HQiw6gV3L_w2B0w2KSTdi6D_7U!1291350239 (visited on 12/03/2023).

[13] Mél Hogan. "Data Center". en. In: *Encyclopedia of Big Data*. Ed. by Laurie A. Schintler and Connie L. McNeely. Cham: Springer International Publishing, 2022, pp. 272–275. ISBN: 9783319320106. DOI: 10.1007/978-3-319-32010-6_299. URL: https://doi.org/10.1007/978-3-319-32010-6_299 (visited on 12/11/2023).

[14] E. A. Huerta et al. "Convergence of artificial intelligence and high performance computing on NSF-supported cyberinfrastructure". In: *Journal of Big Data* 7.1 (Oct. 2020), p. 88. ISSN: 2196-1115. DOI: 10.1186/s40537-020-00361-2. URL: https://doi.org/10.1186/s40537-020-00361-2 (visited on 10/08/2023).

[15] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. "An Efficient Approach for Assessing Hyperparameter Importance". en. In: *Proceedings of the 31st International Conference on Machine Learning*. PMLR, Jan. 2014, pp. 754–762. URL: https://proceedings.mlr.press/v32/hutter14.html (visited on 11/20/2023).

[16] Shoaib Kamil, John Shalf, and Erich Strohmaier. "Power efficiency in high performance computing". In: *2008 IEEE International Symposium on Parallel and Distributed Processing*. ISSN: 1530-2075. Apr. 2008, pp. 1–8. DOI: 10.1109/IPDPS.2008.4536223. URL: https://ieeexplore.ieee.org/document/4536223 (visited on 10/09/2023).

[17] Arvind Kumar et al. “Life Cycle Assessment Based Environmental Footprint of a Battery Recycling Process”. en. In: *Intelligent Manufacturing and Energy Sustainability*. Ed. by A. N. R. Reddy et al. Smart Innovation, Systems and Technologies. Singapore: Springer, 2022, pp. 115–123. ISBN: 9789811664823. DOI: 10.1007/978-981-16-6482-3_12.

[18] Loïc Lannelongue, Jason Grealey, and Michael Inouye. “Green Algorithms: Quantifying the Carbon Footprint of Computation”. In: *Advanced Science* 8.12 (May 2021), p. 2100707. ISSN: 2198-3844. DOI: 10.1002/advs.202100707. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8224424/ (visited on 12/11/2023).

[19] Loïc Lannelongue et al. “GREENER principles for environmentally sustainable computational science”. en. In: *Nature Computational Science* 3.6 (June 2023), pp. 514–521. ISSN: 2662-8457. DOI: 10.1038/s43588-023-00461-y. URL: https://www.nature.com/articles/s43588-023-00461-y (visited on 12/12/2023).

[20] Zhengchun Liu et al. “Characterization and identification of HPC applications at leadership computing facility”. In: *Proceedings of the 34th ACM International Conference on Supercomputing*. ICS ’20. New York, NY, USA: Association for Computing Machinery, June 2020, pp. 1–12. ISBN: 9781450379830. DOI: 10.1145/3392717.3392774. URL: https://doi.org/10.1145/3392717.3392774 (visited on 11/20/2023).

[21] Camilo Mora et al. “Broad threat to humanity from cumulative climate hazards intensified by greenhouse gas emissions”. en. In: *Nature Climate Change* 8.12 (Dec. 2018), pp. 1062–1071. ISSN: 1758-6798. DOI: 10.1038/s41558-018-0315-6. URL: https://www.nature.com/articles/s41558-018-0315-6 (visited on 12/11/2023).

[22] Jeffrey T. Palmer et al. “Open XDMoD: A Tool for the Comprehensive Management of High-Performance Computing Resources”. In: *Computing in Science & Engineering* 17.4 (July 2015), pp. 52–62. ISSN: 1558-366X. DOI: 10.1109/MCSE.2015.68. URL: https://ieeexplore.ieee.org/document/7106398 (visited on 12/28/2023).

[23] Dário Passos and Puneet Mishra. “A tutorial on automatic hyperparameter tuning of deep spectral modelling for regression and classification tasks”. In: *Chemometrics and Intelligent Laboratory Systems* 223 (Apr. 2022), p. 104520. ISSN: 0169-7439. DOI: 10.1016/j.chemolab.2022.104520. URL: https://www.sciencedirect.com/science/article/pii/S0169743922000314 (visited on 11/20/2023).

[24] Simon Portegies Zwart. “The ecological impact of high-performance computing in astrophysics”. en. In: *Nature Astronomy* 4.9 (Sept. 2020), pp. 819–822. ISSN: 2397-3366. DOI: 10.1038/s41550-020-1208-y. URL: https://www.nature.com/articles/s41550-020-1208-y (visited on 11/20/2023).

[25] J. Ranilla, E. M. Garzón, and J. Vigo-Aguiar. "High performance computing: an essential tool for science and engineering breakthroughs". en. In: *The Journal of Supercomputing* 70.2 (Nov. 2014), pp. 511–513. ISSN: 1573-0484. DOI: 10.1007/s11227-014-1279-6. URL: https://doi.org/10.1007/s11227-014-1279-6 (visited on 12/03/2023).

[26] Stephan Schlagkamp et al. "Analyzing users in parallel computing: A user-oriented study". In: *2016 International Conference on High Performance Computing & Simulation (HPCS)*. July 2016, pp. 395–402. DOI: 10.1109/HPCSim.2016.7568362. URL: https://ieeexplore.ieee.org/abstract/document/7568362 (visited on 12/02/2023).

[27] Roy Schwartz et al. "Green ai". en. In: *Communications of the ACM* 63.12 (Nov. 2020), pp. 54–63. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3381831. URL: https://dl.acm.org/doi/10.1145/3381831 (visited on 10/08/2023).

[28] Emma Strubell, Ananya Ganesh, and Andrew McCallum. "Energy and Policy Considerations for Modern Deep Learning Research". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.09 (Apr. 2020), pp. 13693–13696. ISSN: 2374-3468. DOI: 10.1609/aaai.v34i09.7123. URL: https://ojs.aaai.org/index.php/AAAI/article/view/7123 (visited on 10/08/2023).

[29] Nitin Sukhija et al. "Event Management and Monitoring Framework for HPC Environments using ServiceNow and Prometheus". In: *Proceedings of the 12th International Conference on Management of Digital EcoSystems*. MEDES '20. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 149–156. ISBN: 9781450381154. DOI: 10.1145/3415958.3433046. URL: https://doi.org/10.1145/3415958.3433046 (visited on 10/09/2023).

[30] Martin Svedin et al. "Benchmarking the Nvidia GPU Lineage: From Early K80 to Modern A100 with Asynchronous Memory Transfers". en. In: *Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*. Online Germany: ACM, June 2021, pp. 1–6. ISBN: 9781450385497. DOI: 10.1145/3468044.3468053. URL: https://dl.acm.org/doi/10.1145/3468044.3468053 (visited on 10/08/2023).

[31] Kar-Han Tan and Boon Pang Lim. "The artificial intelligence renaissance: deep learning and the road to human-Level machine intelligence". en. In: *APSIPA Transactions on Signal and Information Processing* 7 (Jan. 2018), e6. ISSN: 2048-7703. DOI: 10.1017/ATSIP.2018.6. URL: https://www.cambridge.org/core/journals/apsipa-transactions-on-signal-and-information-processing/article/artificial-intelligence-renaissance-deep-learning-and-the-road-to-humanlevel-machine-intelligence/A82CA4909877C98B23755744A18EA64F (visited on 11/03/2023).

[32] Alex de Vries. "The growing energy footprint of artificial intelligence". In: *Joule* 7.10 (Oct. 2023), pp. 2191–2194. ISSN: 2542-4351. DOI: 10.1016/j.joule.2023.09.004. URL: https://www.sciencedirect.com/science/article/pii/S2542435123003653 (visited on 11/02/2023).

[33] Andy B. Yoo, Morris A. Jette, and Mark Grondona. "SLURM: Simple Linux Utility for Resource Management". en. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 44–60. ISBN: 9783540397274. DOI: 10.1007/10968987_3.

[34] Wucherl Yoo et al. "PATHA: Performance Analysis Tool for HPC Applications". In: *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*. ISSN: 2374-9628. Dec. 2015, pp. 1–8. DOI: 10.1109/PCCC.2015.7410313. URL: https://ieeexplore.ieee.org/abstract/document/7410313 (visited on 10/10/2023).

[35] Qingxia Zhang et al. "A survey on data center cooling systems: Technology, power consumption modeling and control strategy optimization". In: *Journal of Systems Architecture* 119 (Oct. 2021), p. 102253. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2021.102253. URL: https://www.sciencedirect.com/science/article/pii/S1383762121001739 (visited on 02/22/2024).

[36] Zhaobin Zhu, Sarah Neuwirth, and Thomas Lippert. "A Comprehensive I/O Knowledge Cycle for Modular and Automated HPC Workload Analysis". In: *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. ISSN: 2168-9253. Sept. 2022, pp. 581–588. DOI: 10.1109/CLUSTER51413.2022.00076. URL: https://ieeexplore.ieee.org/abstract/document/9912689 (visited on 10/10/2023).