Calculating the Energy Use & Carbon Footprint of Computational Workload Executed on the University of Cambridge National High Performance Computing Service

Adam Calleja
Summer Internship - University of Cambridge Research Computing Service
15/09/2023

Contents

Abstract

The work presented here is the result of an 8 week student internship within the University of Cambridge Research Computing Service. High Performance Computing (HPC) systems such as those deployed at Cambridge consume large amounts of power with the Cambridge system drawing over 1MW of power 24/7 and this is expected to grow rapidly as the system expands. On HPC systems such as this thousands of users run thousands of discrete computational “jobs” on the system per day and environmental sustainability and energy cost control is driving the need for increased energy efficiency of how these computational jobs are executed.

The objective of this project was to pull data from an existing telemetry system and calculate the power consumption and resultant carbon emissions and report them back to users and service management in an easy to see manner. The first step in driving energy efficiency is to be able to identify energy usage and equate it back to individual users and their jobs. The work here was commissioned by DiRAC [1], a large stakeholder group of the Cambridge supercomputer system wanting to drive energy efficiency of their cluster usage, who asked for monthly reports to be sent to their users summarising their energy use & carbon emissions.

The problem was assessed and initial assumptions and simplifications were made to the task to make the project tractable within the short timeframe available. Data analysis was undertaken of SLURM job accounting data for over 2 million jobs run on the Cambridge HPC system. This involved cleaning the data, identifying exclusive jobs, and calculating their resultant carbon emissions. While calculating the carbon footprint of the jobs, we made requests from two APIs: a public carbon intensity API from carbonintensity.org.uk [2] and the Victoria Metrics API containing the node power readings for CSD3. Our framework produces a pandas DataFrame containing the carbon footprint in gCO_2, the energy consumption in Wh, and the equivalent distance travelled by a medium sized diesel car in km for each job that met our initial assumptions.

Although we did not achieve the ultimate goal of sending monthly email reports to users, we created a script that produced a pandas data frame with energy and carbon emissions per job. We then outlined the next steps necessary to implement our pipeline and also made comments to the assumptions and simplifications that could be worked out of the process to produce a more comprehensive job energy report.

During the internship I learned about the architecture and usage of large scale Linux HPC systems, developing problem solving skills with the use of simplifying assumptions. Furthermore, I developed my knowledge of data cleaning and pandas data processing with appreciation of memory usage and multicore processing. Finally the project also required team work and interpersonal skills and the responsibility of working within a large scale production HPC environment. I concluded the project with a presentation that was given to the wider engineering team.

Introduction

The University of Cambridge Service for Data-Driven Discovery, CSD3, is the collection of the two HPC platforms Cumulus and Wilkes3 [4]. Cumulus is a CPU cluster consisting of three partitions of compute nodes of different generations and configurations of Intel CPU's, designed to be used by researchers needing access to HPC facilities [4]. Wilkes3 is a GPU cluster offering "the latest generation high performance computing and AI platform" [4]. These systems have a wide range of different performance and energy use characteristics. The CSD3 system has a wide variety of users ranging from individuals to large organisations such as DiRAC. DiRAC is an organisation which provides distributed HPC services to the Science and Technology Facilities Council, STFC [1]. The DiRAC service pays for power use on the Cambridge system and wanted to maximise power efficiency thus reducing power costs and at the same time drive environmental sustainability.

All users on the Cambridge HPC service execute computational workload onto the HPC system as "jobs". These jobs are submitted to the HPC system via a HPC Batch Job Scheduler called "SLURM" [3]. Jobs are submitted remotely to the SLURM queue where they wait until resources are available to execute the job. The job will then be executed on the system based on various policy and run time conditions. This job scheduling system records a list of all jobs, its user, and where and how the job is executed on the system. The sensible discrete unit of consumption to report on is per job data, where run time metrics can

be measured and reported. Thus the objective of this work is to report the per job energy use and carbon footprint back to the DiRAC users and management to drive energy efficiency.

Before this summer internship project, the University of Cambridge Research Computing Service had no way of calculating the energy consumption or carbon footprint of jobs running on CSD3. As newer HPC systems demand increasing amounts of power to operate, such as the anticipated 7 megawatt consumption projected for Cambridge's forthcoming pre-exascale systems [5], the imperative is maximise energy efficiency. To reduce the energy footprint of HPC clusters it is important that users are aware of both the power and environmental costs of their codes and strive to maximise their codes' efficiency. Additionally, DiRAC have requested that monthly emails be sent to their users providing a summary of their carbon emissions. This initiative aims to promote awareness and encourage users to be mindful of their carbon footprint. As a result, we decided that the focus of this summer internship project would be to develop a framework to calculate and report a user's carbon footprint.

In order to calculate the energy consumption of the CSD3 system, we will be using HPC telemetry data collected from CSD3. The telemetry system in place collects time series data for various targets, which is then stored using Victoria Metrics: an advanced time series database and monitoring solution [6]. Among these targets, our focus lies on the computational nodes for which we will be making power reading requests through the Victoria Metrics' API.

Method

Inspired by DiRAC's request to have monthly carbon reports sent to their users, we established the objective of this summer internship: to develop a framework that calculates the carbon footprint of jobs running on CSD3 and use it to provide users with a summary of their carbon emissions. Before starting the project we had to become familiar with the Linux operating environment of the cluster, its configuration and how to use the SLURM batch scheduling environment. We then had to devise a number of assumptions to simplify the problem before undertaking a data processing pipeline.

Assumptions

During the initial stages of the project, we decided to define some assumptions in order to simplify the problem, so that it was tractable for a project of this short duration.

1. **The jobs run exclusively on nodes.**
   We only have data for the power consumption of each "node" (a node is an individual server within the HPC computer cluster consisting of thousands of servers), rather than individual CPU cores on each node. As a result it would be difficult to determine the precise power consumption of each job, if multiple different jobs share a node. One way to address this problem is to assign each different job the average power consumption across the node for the duration of that job. However, this approach introduces inaccuracies when high power jobs and low power jobs share the same node. A second approach to this problem would be to initially ignore all jobs that

share the same node and assume that jobs run exclusively on nodes. Given that 68.7% of all cluster usage in July consists of such jobs, we opted for the second approach as the most suitable solution, planning to look at non-exclusive jobs later in the project.

2. **The jobs have at least 1 second of runtime.**
   We decided that there is no point in processing jobs which have a runtime of 0s, as they will not produce any carbon emissions.
3. **The jobs have completed.**
   Jobs that have not yet completed will produce more carbon emissions as they continue to run. As a result we decided that it would not make sense to include such jobs in a carbon report, and it would be best to only include completed jobs
4. **We know the number of CPUs per node for each partition.**
   While working on data processing, we used the number of CPUs per node during the identification of exclusive jobs (refer to the 'Processing the Data - Finding Exclusive Jobs' section below for more details). Although this information was readily available for the majority of cluster partitions, obtained through SLURM's *sinfo* command, there were a handful of partitions for which this data was not accessible. Given that only 0.0003% of cluster usage in June had unavailable data, we made the decision to assume knowledge of the number of CPUs per node and subsequently excluded any jobs lacking this data.
5. **A job's energy consumption consists only of the energy consumed by the nodes it runs on.**
   During this project, we only had access to node power readings. Consequently we could only calculate the energy consumption of the nodes themselves, and were unable to account for the energy usage related to cooling, data transfer, or data storage. As a result, we decided to make the assumption that a job's energy consumption consists entirely of the energy consumed by the nodes it runs on.
6. **The maximum runtime of a job is 36 hours.**
   During the identification of exclusive jobs (refer to the 'Processing the Data - Finding Exclusive Jobs' section below for more details) we assume that the maximum runtime of a job is 36 hours in order to maximise the efficiency of our code.

Preparing the Data

The first step to this project was obtaining the data using SLURM's *sacct* command. After experimenting with a couple of jobs that we had submitted ourselves, we decided that we would use the following command to obtain the job accounting data:

> *sacct -Xap --format*
> *JobIDRaw,JobName,Partition,ElapsedRaw,Account,State,CPUTimeRAW,NodeList,User,AllocCPUs,AllocNodes,QOS,Start,End*

Since we did not have access to the job accounting data for all users, we did not submit the command ourselves, but we were given the data as a .csv file. The *'AllocNodes'* column uses commas when multiple nodes are allocated. As a result we could not use a comma as the separator of the .csv file, and opted to use the '|' character instead. We also obtained a .csv file containing the number of CPUs per node for each partition of CSD3 using SLURM's *sinfo* command.

After gaining access to all of the necessary data as .csv files, we read the data into pandas DataFrames to allow for data processing. The job accounting DataFrame contained the raw job ID as the index and the partition DataFrame contained the partition name as the index. The next step was to remove all jobs that did not align with our assumptions:

- We removed all jobs for which the *'CPUTimeRAW'* column had a value of *'0'*, as these jobs would not have run and produced carbon emissions.
- We removed all jobs for which the *'End'* column had a value of *'Unknown'* since these jobs were still running at the time of data processing.
- We removed all jobs for which the value of the '*Partition*' column was not in the index of the DataFrame containing the partition information, since we do not know the number of CPUs per node for these partitions.

After removing all jobs which did not align with our assumptions, we checked the data types of each column of our job accounting DataFrame. In doing so we found that the '*Start*' and *'End*' columns did not contain datetime64 values but rather object values. In order to utilise pandas' full date time functionality, we converted all values in these two columns to datetime64 types using pandas' *to_datetime* function. We then sorted the DataFrame based on the start time of the jobs to allow for easy comparisons to be made between adjacent rows.

Processing the Data - Finding Exclusive Jobs

The next step of the summer internship project was the identification of *exclusive jobs*. During this project, we defined an exclusive job as follows:

> *An exclusive job is any job which does not share the nodes it runs on with any other jobs during its runtime.*

Furthermore, we defined two mutually exclusive categories of exclusive jobs:

1. **Jobs which are exclusive by CPU count.**
   These are jobs which have been allocated all of the CPUs on the nodes that they are using.
2. **Jobs which are exclusive by loneliness (lonely jobs).**
   These are jobs which are allocated only a part of a node, but do not overlap with any other jobs running on that node.

We began by identifying the jobs which are exclusive by CPU count. Using the partition data we had obtained from SLURM, we compared each job's total allocated number of CPUs across all allocated nodes to the maximum number of CPUs which could have been allocated for that partition. The jobs for which these two values are equal are exclusive by CPU count. After identifying all jobs which fit into the first category of exclusiveness, we needed to find all jobs which are exclusive by loneliness. Since the two categories are mutually exclusive, we first filtered the DataFrame to remove all jobs which are exclusive by CPU count, thereby minimising the amount of data needed to be processed. To find lonely jobs, we need to compare the start and end times of every job that runs on the same node. If

two jobs run at the same time, then they are not lonely. Before comparing the start and end times of jobs, we segregated the multi-node jobs into distinct rows for each node they were allocated to. This approach enabled us to compare each job running on a node and identify any overlapping instances. We then created a function which returns a list of all the jobs that overlap with a given job. The function takes in two parameters: the row corresponding to the given job and the job accounting DataFrame the row is from. In order to maximise the efficiency of our code, the function only compares the given job to jobs which start at most 36 hours before the given job starts and end at most 36 hours after the job ends. After testing the function on a fabricated set of data containing seven jobs and ensuring that it worked as expected we were ready to apply our function to the job accounting dataset. In order to further improve the efficiency of our code, we decided to implement parallelisation using the *joblib* library. To accomplish this, we divided the job accounting DataFame into separate DataFrames, each corresponding to a specific node. This division streamlined the process of comparing jobs with others on the same node, eliminating the necessity to compare them to jobs running on different nodes. Subsequently, we implemented our function, utilising the joblib library with 8 cores, which yielded a list of DataFrames that we later concatenated. To ensure our code had worked, we manually checked if the results for a random node were as expected. We then added three new columns to our original job accounting DataFrame: one containing whether or not the job is exclusive by CPU count, one containing whether or not the job is exclusive by loneliness, and one containing whether or not the job is exclusive by either definition.

During analysis of the data, we found that 68.7% of cluster usage in July was made up of exclusive jobs, all of which are exclusive by CPU count. Furthermore we found that 41% of shared cluster usage consists of jobs submitted by the same user.

Processing the Data - Calculating the Carbon Footprint

The final phase of the summer internship project involved calculating the carbon footprint. To calculate the carbon footprint of the jobs run on CSD3, our initial step was to query Victoria Metrics to acquire power readings for the nodes. The first problem that we encountered was that while Victoria Metrics used times in UTC, SLURM's sacct command returned times in the local timezone (BST at the time of writing). To overcome this issue, we created a function which converted all times in the DataFrame from the local time in *Europe/London* to UTC. While devising our approach, we initially considered the possibility of querying Victoria Metrics once at the outset and storing all the power data for all nodes over the specified timeframe in a .csv file, as an alternative to repeatedly querying Victoria Metrics for each individual job. However, we found that this approach uses a lot of RAM because pandas stores DataFrames in memory rather than on the disk. As a result, this approach proved to be inefficient, leading us to ultimately choose the method of querying Victoria Metrics to obtain the power readings for each job individually.

We opted for creating separate functions for each step of the process, which we then applied at the end in order to obtain the final carbon footprint. Our first function queried the Victoria Metrics API, obtaining the power readings for the nodes the job runs on. This function accepts two parameters: the job ID and the job accounting DataFrame. It returns a DataFrame with the timestamp as the index and columns representing the power readings, measured in watts (W), for each node the job runs on. In the event of an API issue, the

function returns *None*. The second function calculates the energy consumed by each node job runs on, in watt hours (Wh), for each 30 minute period of the job's duration. It takes in the DataFrame returned by the first function and returns a DataFrame with the 30 minute period as the index and the columns representing the energy consumption of each node. We opted to separate the energy consumption into 30-minute segments to enhance the precision of our results. The decision was made because the carbon intensity API [2] provides distinct carbon intensity values for each 30-minute segment of the day. In contrast to our previous approach to querying the Victoria Metrics API, we decided that the best approach to querying the public carbon intensity API [2] is to query it initially and store all of the carbon intensity data in a .csv file which we can access when needed. We made this decision since we did not want to make a large amount of requests to the public API for each job we are processing. Finally, we created a function to calculate the carbon footprint of a given job. This function takes both the job ID and the job accounting DataFrame as parameters. It calls the functions created beforehand and calculates the carbon footprint, measured in grams of carbon dioxide (gCO_2), using the equation below:

$$carbon\ footprint\ =\ energy\ \times\ carbon\ intensity$$

The function returns a DataFrame containing the job's energy consumption, measured in watt hours (Wh); the carbon footprint, measured in grams of carbon dioxide (gCO_2); and the equivalent distance travelled by a medium sized diesel car, measured in kilometres (km). After validating the functions on a set of fabricated power readings, we proceeded to apply them to the job accounting DataFrame, leveraging parallelism through the utilisation of the joblib library. This resulted in 3 new columns being added to the job accounting DataFrame representing the 3 values returned by our final function.

Output from the Project

The final output of this summer internship project is a Python script which takes in three parameters from the command line representing the paths to the job accounting, partition information and the final .csv files. This python script adds 3 new columns to the input job accounting DataFrame representing each job's energy consumption, measured in watt hours (Wh); carbon footprint, measured in grams of carbon dioxide (gCO_2); and the equivalent distance travelled by a medium sized diesel car, measured in kilometres (km). The script then writes this final DataFrame to a .csv file at the path specified in the third input parameter. Although we did not meet the ultimate end goal of reporting users' carbon emissions, we successfully developed a framework that calculates the carbon footprints of jobs running on CSD3, which can be implemented easily and produce energy and carbon emission data within the limitations of the assumptions that were made.

Future Work

While our project fell short of its initial goal of providing users with reports of their carbon footprint and energy usage, we have identified the necessary steps required to bring our project's objectives to realisation. The first step would be to run the scripts we have created

every month and then send an energy usage and carbon footprint report to users. This would involve the following substeps:

1. Create an email template to send to users.
2. Write a python script to send consistent email reports to users.

This would provide to the Cambridge Research Computing Service and its users and stakeholders a good high level overview of energy use and carbon emissions on a per job basis so that users could think about how to make their computational jobs more efficient.

There were several simplifications introduced in order to make the project tractable within the short 8 week period of the internship. Additional future work could be undertaken to remove these simplifications. One limitation of our Python script is that it only calculates the carbon footprint of exclusive jobs. The first step to address this problem would be to include jobs which only share nodes with other jobs that are submitted by the same user. During the analysis of the job accounting data from July, we found that 41.5% of shared jobs would fall into this category. We could then treat each group of shared jobs as one exclusive job, reporting a single carbon emission for each distinct group of shared jobs. The next step to address this problem would be to take all shared jobs into account, reporting an estimate of their carbon emissions. One further limitation of our implementation is that we have only taken into account the energy consumption of the computational nodes. To mitigate this limitation we could consider the energy consumption due to cooling, data transfer and data storage.

Skills learned

During the project many useful skills were developed:

- Developed knowledge of large-scale Linux-based HPC systems.
- Gained proficiency in Linux HPC user environments.
- Acquired skills in using the SLURM batch scheduling system for job submissions.
- Learned to analyse complex projects and implement simplifications and assumptions for tractability.
- Recognised the impact of simplifications on project outcomes.
- Demonstrated proficiency in data analysis, including initial data cleaning.
- Applied efficient data processing techniques using pandas.
- Utilised concepts like memory management and multicore processing.
- Enhanced communication skills by presenting project findings to a broader engineering team.
- Collaborated efficiently within a team environment.
- Managed a large-scale production system responsibly.

Conclusion

Throughout this summer internship project, we achieved the successful development of a framework for calculating the carbon emissions generated by exclusive jobs running on

CSD3. While we were in close proximity to achieving our ultimate goal of sending users reports detailing their carbon emissions, it remains crucial to acknowledge and address the limitations inherent in our current implementation, as previously discussed. Clear future work has been mapped out and discussed with the team so this initial work can be implemented and then extended.

In addition to creating a useful first implementation calculating per job energy use and carbon footprint the project was highly successful in training an undergraduate student in the architecture and usage of a large-scale Linux HPC system, developing problem solving skills with the use of simplifying assumptions and finally intermediate pandas data processing skills with an appreciation of memory usage and multicore processing. The project also required team working skills and the responsibility of working within a large-scale production HPC environment.

Acknowledgments

Many thanks to Projeet Bhaumik, a fellow student Intern who helped with the initial processing and analysis of the job accounting data and Dominic Friend who mentored me throughout this summer internship project and was extremely helpful and supportive.

References

[1] DiRAC. DiRAC High Performance Computing Facility – Supporting the STFC theory community. dirac.ac.uk. Available at: https://dirac.ac.uk/ (Accessed 15/09/2023)
[2] National Grid. Carbon Intensity API. carbon-intensity.github.io. Available at: https://carbon-intensity.github.io/api-definitions/#carbon-intensity-api-v2-0-0 (Accessed: 21 August 2023)
[3] SchedMD. Slurm Workload Manager - Overview. slurm.schedmd.com. Available at: https://slurm.schedmd.com/overview.html (Accessed 15/09/2023)
[4] University of Cambridge. High Performance Computing. hpc.cam.ac.uk. Available at https://www.hpc.cam.ac.uk/high-performance-computing (Accessed 15/09/2023)
[5] University of Cambridge. Energy Efficient Supercomputing | Research Computing Services. hpc.cam.ac.uk. Available at: https://www.hpc.cam.ac.uk/energy-efficient-supercomputing (Accessed 15/09/2023)
[6] Victoria Metrics. Open Source Time Series Monitoring Tools & Solutions. victoriametrics.com. Available at: https://victoriametrics.com/products/open-source/ (Accessed 15/09/2023)