Strategies to Measure Energy Consumption Using RAPL During Workflow Execution on Commodity Clusters

Philipp Thamm

May 15, 2025

Abstract

In science, problems in many fields can be solved by processing datasets using a series of computationally expensive algorithms, sometimes referred to as workflows. Traditionally, the configurations of these workflows are optimized to achieve a short runtime for the given task and dataset on a given (often distributed) infrastructure. However, recently more attention has been drawn to energy-efficient computing, due to the negative impact of energy-inefficient computing on the environment and energy costs. To be able to assess the energy-efficiency of a given workflow configuration, reliable and accurate methods to measure the energy consumption of a system are required. One approach is the usage of built-in hardware energy counters, such as Intel RAPL. Unfortunately, effectively using RAPL for energy measurement within a workflow on a managed cluster with the typical deep software infrastructure stack can be difficult, for instance because of limited privileges and the need for communication between nodes. In this paper, we describe three ways to implement RAPL energy measurement on a Kubernetes cluster while executing scientific workflows utilizing the Nextflow workflow engine, and one additional method using IPMI. We compare them by utilizing a set of eight criteria that should be fulfilled for accurate measurement, such as the ability to react to workflow faults, portability, and added overhead. We highlight advantages and drawbacks of each method and discuss challenges and pitfalls, as well as ways to avoid them. We also empirically evaluate all methods, and find that approaches using a shell script and a Nextflow plugin are both effective and easy to implement for workflow users. Additionally, we find that measuring the energy consumption of a single task is straight forward when only one task runs at a time, but concurrent task executions on the same node require approximating per-task energy usage using metrics such as CPU utilization.

1 Introduction

During scientific work, a series of computational steps is often required to extract the desired information from a given dataset. Such pipelines of multiple independent programs for data analysis are commonly referred to as scientific workflows [25]. Scientific workflows are used in many areas, such as genomics [11] or remote sensing [35]. Most of the time, workflows are being optimized for fast execution speeds in order to reduce the time needed for data analysis. However, the increasing awareness of the growing greenhouse gas emissions of the Information and Communication Technology (ICT) sector, which already accounted for up to 2.8% of all global emissions in 2022 [12], emphasizes the need for energy-based workflow optimization.

To be able to optimize a workflow for energy-efficient execution, developers need access to information regarding the current energy consumption of their workflow. One way to achieve that is the utilization of built-in Intel RAPL energy counters [28, 19, 7]. While accessing these energy counters is relatively straightforward on local hardware through available software such as perf[1] or IPMI[2], it can pose a significant challenge in more complex environments. With the help of these energy counters, the energy consumption of CPU, integrated graphics, I/O-controllers, cache and RAM can be tracked depending on the CPU model [19], leading to estimations of the energy consumption which can be utilized to predict the energy consumption of the whole system [17].

In this paper, we present our lessons-learned from implementing multiple methods to read the RAPL energy counters while executing scientific workflows on a shared, managed commodity cluster using the Nextflow workflow management engine [34] and Kubernetes [33]. We discuss the challenges of reading RAPL values posed by the underlying infrastructure and present several

[1] https://perfwiki.github.io/main/, last accessed: May 14, 2025

[2] https://www.intel.de/content/www/de/de/products/docs/servers/ipmi/ipmi-home.html, last accessed: January 10, 2025

solutions, considering a setup that leverages Nextflow and Kubernetes. Additionally, we provide an overview of the technologies employed, as well as a perspective on potential future work to facilitate the use of the information provided by RAPL.

In the following, we provide some background about the used technologies in Section 2. Then, we outline the general circumstances leading to problems with the automated measurement of energy consumption via RAPL in our setup in Section 3 before presenting multiple ways for automated energy measurement despite these pitfalls in Section 4. Our experimental results are presented in Section 5. This is followed by a discussion of our findings in Section 6. Section 7 highlights opportunities for future work. Finally, we conclude our work in Section 8.

2 Background

This section discusses the most important technologies related to scientific workflows (2.1), energy measurement (2.2) and especially RAPL (2.3), as well as workflow infrastructure (2.4).In the context of scientific workflows on compute clusters, several key terms must be clearly distinguished. For this work, a cluster is a set of independent machines working together to execute computations efficiently. Each machine in the cluster is called a node, which is an independent computer capable of executing tasks. Nodes contain at least one CPU, which itself consists of multiple cores that can handle separate workloads simultaneously. A task is a distinct executable computational step within a workflow, defined by its inputs and outputs, and can be reused across workflows.

2.1 Scientific Workflows

The computational analysis of large amounts of scientific data is often complex, leading to significant computational requirements. Many analysis pipelines consist of multiple interdependent steps, known as tasks. These pipelines of linked, interdependent data analysis tasks are called scientific workflows [14]. Due to the large inputs, high computational requirements and the segmentation into interdependent tasks, which can often be executed in parallel, scientific workflows are often executed on compute clusters consisting of multiple compute nodes (Section 2.4).

When composing a scientific workflow, researchers define the inputs, workflow steps, and their dependencies. They use workflow languages to express this information in a format that the executing machine can interpret. These workflow languages are often embedded into workflow management systems, programs designed to facilitate the composition and implementation of workflows [25]. Additionally, they also help to make workflows more reproducible. Workflow management systems have varying feature sets. For instance, Galaxy [36] provides a graphical user interface (GUI) for the implementation and execution of workflows, improving its accessibility. Other workflow management systems like Snakemake [26] or Nextflow [8] use domain specific languages (DSL) to enable workflow implementation. Due to utilizing a DSL, the code written in these workflow management systems might be easier to write and read compared to general programming languages, since the features of the language are specifically tailored to scientific workflows. The use of a textual workflow language also improves portability and enables easy versioning through version management tools like Git.

Nextflow

Nextflow [8] is a popular workflow management system primarily used in the area of Bioinformatics. Nextflow workflows are written in Groovy, a DSL based on Java. It offers support for containerization and orchestration technologies such as Docker (Section 2.4) and Kubernetes (Section 2.4), among others. For this reason and due to the availability of multiple different workflows from the area of Bioinformatics, we chose to use Nextflow for our experiments.

Optimizing Workflow Execution

Due to the complex structure of scientific workflows consisting of multiple interdependent tasks, optimization to a specific infrastructure is not trivial. Possible goals of optimization are minimal makespan [2] (the total execution time of the workflow from beginning to end), or minimal memory utilization for each individual task [21]. Sometimes, approaches aim for multi-objective optimization [18]. Other situations require in-budget optimization, where a workflow is optimized while adhering to specific constraints [22]. Another possible objective is the reduction of a scientific workflows energy consumption [10]. To minimize the energy consumption of scientific workflows and individual workflow tasks, researchers need access to data regarding the energy consumption

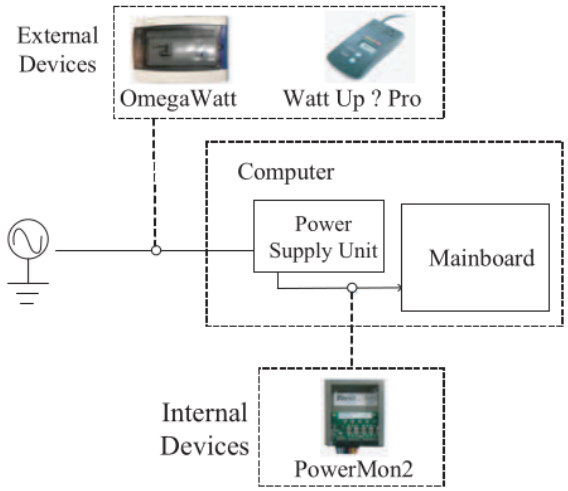

Figure 1: The installation locations for external and internal physical power meters [23].

of the current configuration. Therefore, researchers require reliable methods to measure the task energy consumption of a workflow.

2.2 Methods of Energy Measurement

Physical Measurement

The first method to measure the power consumption of a device is through the use of an external power meter. These measurement devices are generally positioned between the wall socket and the power supply of a device or computing node. They measure the total power consumption of the entire computing node. Since the entire power flowing to the node also flows through the power meter, these devices are typically very accurate [16]. They also have the advantage that their impact on the measured system is negligible. On the other hand, installing power meters requires an additional financial investment, and it is not possible to individually measure the power consumption of concurrently running tasks or specific hardware components such as CPU, DRAM, disk or power supply. Additionally, this approach to measuring power is impractical for large systems, since an additional power meter is required for each node.

Alternatively, intra-node power meters can also be used for physical measurements. Unlike external power meters, these devices are placed *inside* of a computing node. They can be placed between the power supply and the main board, such as PowerMon2 [4], or next to specific components, such as PowerInsight [20]. Intra-node power meters can measure the power consumption of specific components such as CPU, GPU or cooling fans, but they share many drawbacks with external power meters. In addition, they are often expensive to install and inflict additional liability issues. Figure 1 shows typical installation locations of external and internal physical power meters on a computer.

Approximation through Energy Models

Energy models use a set of metrics collected during or after the program execution to calculate an approximation of the energy consumption. These metrics can include CPU frequencies, run time, the thermal design power of the processor, the number of cache misses, fan speed and many more. Based on these metrics, numerous formulas to calculate task power consumption have been proposed [23]. An advantage of energy models is that they do not require any additional hardware or the existence of specific feature sets like hardware energy counters. The most important disadvantage of software-based energy models is that it is hard to define a metric that works well across different machines, architectures and software [27]. One model that performs well for one task on a specific machine might perform much worse when used in a different context, making it much less useful when applied outside the context it was originally optimized for.

Hardware Energy Counters

Most component manufacturers embed digital sensors or onboard measurement circuits to measure the power consumption of the entire system, the CPU cores, the memory or other components in

their products.

Intel introduced the Running Average Power Limit (RAPL) in 2011 [7] in the Sandy Bridge architecture as a software power model based on architectural events including the cores, integrated graphics and I/O. In the Haswell architecture, this implementation was exchanged with a version based on fully integrated voltage regulators, enabling actual power measurement and improving the accuracy of RAPL [15]. More information about Intel RAPL is presented in Section 2.3.

AMD introduced a version of hardware energy counters similar to the first version of RAPL with their Zen architecture. Similar to the early version of RAPL, this implementation can provide inconsistent results due to using execution path-based modeling instead of on-chip energy measurement [31].

NVIDIA provides users with an API called NVIDIA Management Library (NVML) that provides GPU device metrics such as current utilization, temperature and power draw [32].

The advantage of hardware energy counters is the typically good accuracy compared to purely software based models due to the deep integration in the system components and the utilization of integrated measurement hardware [15, 31]. One drawback is that only the system components equipped with appropriate measurement hardware by the hardware designer can be measured. For example, RAM and the on-chip I/O-controllers are measurable in some implementations of Intel RAPL [19], but other components such as secondary storage (e.g., disks or solid state drives) and interconnecting network are not equipped with appropriate energy counters, and therefore can not be measured. Additionally, available interfaces and domains for energy measurement based on hardware energy counters are vendor-specific and can vary between models. This limits the portability of software utilizing hardware energy counters.

2.3 Running Average Power Limit

Running Average Power Limit (RAPL) from Intel is an interface to estimate power usage that is built into many Intel CPUs [7, 15]. It was first implemented as a software power model in 2011 [7] in the Sandy Bridge architecture. Starting with the Haswell architecture, it was changed to using fully integrated voltage regulators, making actual power measurement instead of estimations possible and thereby improving the accuracy of the results [15].

RAPL is capable of reporting the energy consumption in different power domains. Figure 2 shows an overview of the supported domains and which parts of the system are contained in each of them.

- The Core domain (here labeled Powerplane 0) reports the energy consumption of all CPU cores added together. Note that RAPL is not capable of reporting the power usage of an individual CPU core.
- The Graphics domain (Powerplane 1) reports the energy usage of the integrated graphics component. If the chip has an eDRAM component (a small amount of RAM embedded directly on the chip, acting akin to a Layer 4 cache), its power consumption is also included in the Graphics domain.
- The Package domain contains both the CPU cores and the integrated graphics, in addition to some components not contained in the Core or Graphics domain. This includes higher level caches which are not directly associated with a CPU core, the memory controller, system agent and the I/O controller (not shown).
- The DRAM domain is separate and reports the power usage of all the DRAM modules added together. It is important to note that the power used by the DRAM is not included in any other domain. This makes the DRAM domain completely separate of all the other domains, and the only domain that logs exclusively the power consumption of components not located on the CPU chip.
- The Psys domain reports the power consumption of almost the whole system. It includes the Package domain and most additional components on the CPU, as well as some additional system components such as cooling fans. Notable exceptions are the DRAM, which is exclusively monitored by the DRAM domain, and secondary storage like disks, which is not monitored by RAPL.

Not all RAPL domains are available on all CPUs, but the availability of each domain depends on the microarchitecture and individual model of CPU. Documentation about each model of CPUs

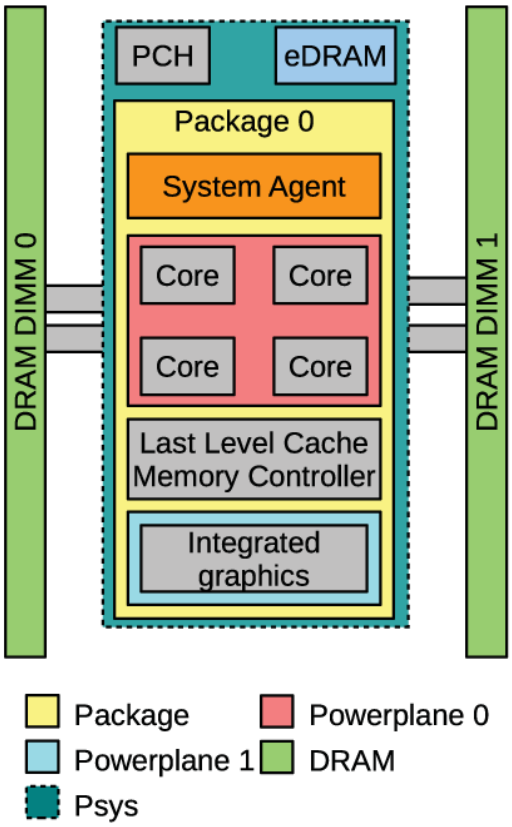

Figure 2: An overview of the system components covered by each RAPL domain [19].

is available in the respective specifications on Intel's website[3].

RAPL energy consumption reports can be accessed through Model-specific registers (MSR). The values in these registers are expressed in energy units and represent the energy consumed in microjoules (μJ) since the processor was started, multiplied by a static conversion value that depends on the domain and the individual model of CPU. Each MSR that contains the energy counter for one of the domains is updated roughly every 1 ms (1000 Hz) with low overhead [19]. The RAPL power monitoring is always running, starting when the machine is booted up. The machine-specific registers (MSRs) are limited in their number of bits. Some registers are limited to 32 bits, others to slightly larger values, e.g., 36 bits, depending on the CPU model. Therefore, each energy counter occasionally overflows and loops around to zero. This is not logged or communicated in any way. For example, an Intel(R) Xeon(R) Silver 4314 CPU completed a loop from zero to overflowing in 52 minutes while executing a workload of mixed intensity during our experiments. Therefore, measuring the energy over a specific time frame requires continuous monitoring of the MSR to catch overflows and factor them into the calculated energy value. Additionally, timestamps are not automatically attached to the read values.

In terms of general accuracy of energy measurement conducted using Intel RAPL, earlier research concludes that the energy consumption reported by RAPL is reliable for both CPU [16] and memory [1]. Currently, we can not confirm if this is also the case for scientific workflows executed on compute clusters, since we do not have access to physical power meters for accurate power readings on the cluster hardware.

2.4 Workflow Infrastructure

Clusters are distributed systems [37] that allow for fast execution of tasks by providing computational resources beyond what a singular machine can offer. A cluster consists of a number of independent computing elements, called nodes. Each node is a complete system with at least its own CPU and memory. These nodes are interconnected with each other to enable communication and data transfer. Usually, one of the nodes in the cluster functions as the management node, controlling and scheduling the worker nodes. The user communicates with the cluster through

[3]https://www.intel.com/content/www/us/en/ark/products/series/122139/intel-core-processors.html, last accessed: May 14, 2025

the management node, which makes the whole cluster appear as a singular structure to the user. To make this form of interaction of the user with the cluster possible, an orchestration system is needed to schedule available resources of the cluster transparently and automatically. Kubernetes [5] is a widely used container orchestration system that is often used to coordinate the execution of workloads, including scientific workflows, on compute clusters.

Kubernetes

Kubernetes [5] is a container orchestration system that was originally created by Google. It was donated to the Cloud Native Computing Foundation in 2016 and is now open source. In Kubernetes-managed environments, a pod represents an execution unit that is assigned to a node with a defined amount of resources (CPU, memory). Typically, each pod executes a single task, except for a dedicated command-and-control pod, which orchestrates workflow execution. Each pod encapsulates a container, which contains the program along with all its dependencies, ensuring a consistent execution environment for a task. Kubernetes can be used to deploy, monitor and scale containers on a compute cluster or other distributed architectures. It implements the automatic scheduling and re-scheduling of containers on a distributed infrastructure, and also supports automatic resource allocation and load balancing. These features make Kubernetes very useful when workflows are executed on a compute cluster. Kubernetes can work with multiple types of containers. These containers are often managed by Docker [30], a platform for sharing and managing containers.

Docker

The containers utilized by Kubernetes are built and managed by Docker [30]. Docker is an open platform to package, distribute and deploy applications in a secure, portable and lightweight manner. It allows users to package applications into Docker images, together with all their needed resources and environments. These Docker images can then be shared and run on other machines with no additional configuration through the Docker Engine. This process is much more lightweight than virtualization [30]. At the same time, Docker images run faster and similarly secure since each container runs in its own virtual environment with no access to resources outside of it. These characteristics make Docker containers ideal for executing the tasks of a workflow. They allow the user to deploy different applications used in the same workflow independently on any node in the cluster, sequentially or in parallel. At the same time, they are secure and no additional configuration on the nodes is necessary.

3 Challenges for Automatic Measurement of Energy Consumption of Scientific Workflows Executed on Compute Clusters

To determine the energy consumption of a machine over a timeframe, one can use RAPL energy counters. The counter values can be read from the respective MSRs and written to a text file. Time stamps are manually added when writing new energy counter values to the file in order to make correlation of the values with a specific point in time possible. This way, it is possible to calculate the energy consumption of the machine over any period of time by calculating the difference between the values stored in the energy counters at the starting point and at the end of the time period.

In order to measure the energy consumption of a scientific workflow, the RAPL values are collected right before the workflow is started and right after the workflow has finished. By subtracting the value of the energy counter before execution from the value of the energy counter after execution, the amount of energy consumed by executing the workflow can be calculated, provided that the workflow was the only workload executed on the node during measurement. Task-wise energy measurement is also possible by the same method (reading RAPL energy counters at the start of the task and when the task is finished), but again only if the task is the only workload executed on the machine. If multiple workloads are executed on the same machine in parallel (e.g., multiple tasks from the same scientific workflow or from two workflows running concurrently), an additional heuristic such as the CPU time used by each of the tasks must be applied, because RAPL only captures the total energy consumption over the time period.

The fact that the workflows are run on a cluster consisting of multiple independent machines orchestrated by Kubernetes leads to some inherent complications for power measurement using

RAPL when compared to a local execution. In this section, the issues arising and possible steps to be taken are discussed.

3.1 Multiple Nodes

The workflows are executed on a cluster consisting of multiple independent nodes with their own hardware, including the CPU. It is not clear in advance which and how many nodes of the cluster will be used during the execution of the workflow. The user only has a limited influence on the scheduling in Kubernetes by specifying parameters such as the utilized namespace or the number of splits of a data set, leading to more or less parallelism. Therefore, the energy consumption of each node that could *potentially* participate in the execution of *any* part of the workflow needs to be measurable. However, RAPL is only capable of measuring the power consumption on the same local chip, but not remotely for other CPUs. That means that the power consumption of each CPU in each node that could potentially be used during the workflow execution must be prepared for measurement. If a workload (i.e., one of the tasks) belonging to the measured scientific workflow is executed on any of the machines, its energy consumption must be measured over that time period. After workflow execution has finished, the power consumption of all the involved nodes has to be added to calculate the total energy consumption.

3.2 Required Privileges

The program measuring energy consumption requires root privileges to access the values stored in the RAPL registers. This is necessary because the fine-grained energy measurements provide a vector for side-channel attacks by closely monitoring the consumed energy and reconstructing the executed instructions from these measurements. It has been proven that such attacks are feasible [24]. For instance, DeepTheft [13] is a tool designed to steal Deep Neural Network weights through an Intel RAPL side-channel attack.

For this reason, one can not access the RAPL registers from the typical container process running on the cluster. Instead, a container with the required privileges to read RAPL registers must be deployed on each utilized node.

3.3 Constrained Docker Containers

In a Kubernetes setup, each task of the workflow, including the main control task, are running in isolated Docker containers, which means that the tools at their disposal are limited. The typically very small Docker containers often lack even basic system tools, simply because they are not needed for the task performed in the container. For example, the vi text editor, normally included with any Unix-based operating system, is sometimes missing in small Docker containers. This reduces the amount of options available to control the process of RAPL measurement without deploying custom containers with added tools.

3.4 Requirement of Continuous Measurement

Another factor that complicates the process of obtaining accurate energy values from RAPL energy counters is that a single read-out before and after each task of the workflow is not sufficient. The used power is represented in the RAPL register as energy units in microjoule (μJ) [19]. Due to the small unit being utilized and the register being limited to 38 bits, the number in the register overflows to zero between every half an hour up to every 24 hours, depending on the computational requirements of the workload. These overflows are not logged or indicated in any way. To avoid reporting wrong values due to an undetected overflow, it is necessary to constantly monitor the RAPL register in sufficiently small time intervals to detect overflows, so they can be considered when calculating the total amount of used power.

3.5 Available RAPL Domains

Not all RAPL domains presented in Section 2.3 are available on every CPU. For instance, the Intel(R) Xeon(R) Silver 4314 CPUs running in the cluster used for this research support only the package and DRAM domains. Fortunately, these two RAPL domains contain almost all the components that are included in any of the other RAPL domains, providing an almost complete picture of the energy consumed by the components monitored by RAPL. These components account for about 63% of the total energy consumed by a typical server [23], and up to 79% if the average

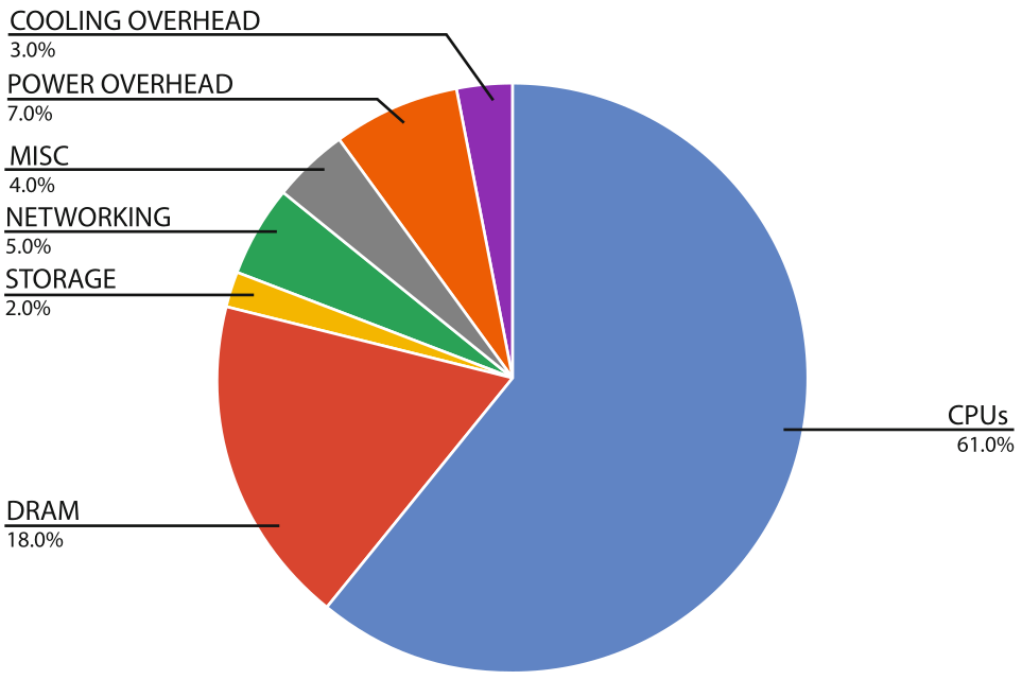

Figure 3: A breakdown of the typical power consumption of a physical server [3]. The figure assumes two-socket x86 servers and 12 DIMMs per server, and an average utilization of 80%.

utilization is high [3]. The other 21% are consumed by the disk (2%), the network (5%) and other parts of the system (14%) such as cooling fans or the power supply. Figure 3 shows a breakdown of the typical power consumption of a server with an average utilization of 80%.

3.6 Granularity

To be able to use the data collected during energy measurement for workflow optimization, it is important to measure with high granularity. Here, one needs to differentiate between workflow granularity and hardware granularity. In terms of workflow granularity, it would be ideal to measure the energy consumption of the whole workflow, of each physical or abstract workflow task, and of each individual container that is part of the workflow. From the hardware perspective, the goal is to measure the energy consumption of each node, each individual CPU core on a node, and each individual thread running on a core. In practice, the limiting factor is the availability of RAPL domains. If the only available domain is the Package domain, it is not possible to measure individual cores or threads, but only the energy consumption of the whole CPU for each individual CPU in the system. From the workflow perspective, this means that the energy consumption of a workflow, task or container can only be individually measured if no other software is running on the same machine at the time of execution. If some other software is running at the same time as the workflow, task or container, the energy consumption can only be estimated by subtracting the energy consumed by that other software utilizing a separate metric.

4 Measurement Strategies

In this section, we present four measurement strategies we developed in order to achieve reliable and accurate energy measurements for workflow executions in Nextflow on a Kubernetes cluster utilizing Intel RAPL and IPMI. We explain each strategy in detail and discuss their respective advantages and drawbacks.

4.1 Design Goals for Energy Measurement

We identified a set of eight characteristics an ideal approach for RAPL-based energy measurement of workflow tasks should have.

No additional software necessary

An ideal approach for RAPL-based energy measurement works without additional software that needs to be installed and maintained separately on the cluster by the cluster-administrator or the workflow-administrator. Depending on additional software makes an approach for measurement harder to configure and limits the scope where that method can be applied, since the needed software might not be available when working with different clusters or workflow systems.

Self-contained on the cluster

Approaches for energy measurement should not require a continuous connection to an external device from the cluster, such as the user's local machine. If such a connection is necessary, it limits how the user can utilize their machine while the workflow is running on the cluster. Most importantly, the local machine would need to stay active and connected to the network in order to ensure that continuous communication between the local machine and the cluster is possible. This is a significant disadvantage in usability, especially for long-running workflows where the user might want to turn off their local machine or leave the network, like for executions of workflows overnight.

Dealing with workflow faults

Software for energy measurement should be able to react to faults or unexpected changes in the number and order of executed workflow tasks, and still be able to measure the energy consumption of each task correctly. Otherwise, unexpected events during workflow execution might lead to incomplete or wrong energy consumption data.

Full measurement for all workflow tasks

An ideal approach for RAPL-based energy measurement should be capable of capturing the full energy consumption individually for all physical tasks in a workflow. The measurements for the physical tasks can then be aggregated to represent the energy consumption of logical tasks or the whole workflow. If the energy data for some tasks is incomplete, its usefulness for future usage will be limited.

Easy portability to other workflows and workflow systems

Implemented approaches for energy measurement should be workflow-agnostic and easy to implement for a new workflow. Ideally, they are also portable to other workflow systems to enable broad usage.

No workflow modifications necessary

Enabling energy measurement for a workflow should require no changes to the structure or implementation of the workflow. Such requirements would make implementation of compatible workflows more tedious and difficult, and therefore limit the usability of the tools for measurement.

Low overhead

Every approach for RAPL-based energy measurement will add *some* computational overhead during workflow execution. This computational overhead will cause additional load on the system and increase the energy consumption. It is important that the extent of this overhead is kept small and limited to the time of workflow execution.

Multi-tenancy

Methods for RAPL-based energy measurement should be capable of measuring the energy consumption of multiple independent scientific workflows running in parallel on the same nodes of the cluster.

4.2 Approaches

In the following, four approaches to monitor workflow energy consumption by reading values from RAPL energy counters are presented, focusing on Nextflow as workflow management system and Kubernetes as container orchestration system. Except for the method using the API of Prometheus, all methods use dedicated pods, running on each node of the cluster, that were configured with special permissions to be able to read from RAPL registers. When receiving a command, these pods execute a script to continuously read the RAPL energy counters and store the values in log-files together with time stamps. These log files are stored on the shared storage of the cluster and can be accessed by any pod. As soon as a physical task is finished, its energy consumption can be calculated either directly on the cluster or on the local machine of the workflow user by copying the file. When the workflow is completed, its energy consumption can be calculated by summing up the values for all physical tasks. When using Prometheus, the energy consumption does not have to be calculated manually. Prometheus automatically converts the measured values into consumed energy. The consumed energy for a specific time interval can be directly requested using the API. Within this general setting, the four methods differ in how they coordinate the process of measurement. Since reading and logging RAPL values continuously would waste valuable system resources, including CPU overhead and required storage, it is more efficient to only measure RAPL values when they are needed. The proposed methods offer different strategies to measure RAPL counters only during workflow execution, or otherwise circumvent the resource overhead introduced by continuous measurement.

Table 1 shows which of the eight design goals for energy measurement are fulfilled by each of the four approaches. A checkmark ("✓") means that the criterion is fulfilled completely, while the cross ("✗") denotes a criterion that is not fulfilled. A checkmark in brackets ("(✓)") symbolizes a criterion that is technically not completely fulfilled. For example, the Plugin method technically requires the plugin as additional software. However, if the plugin was officially released, it can be integrated into a Nextflow workflow using one line of code or one command line argument. It is automatically downloaded and installed during workflow execution and therefore does not cause any additional work for the workflow developer or the workflow user. For this reason, the criterion can be considered fulfilled.

Table 1: Comparison of Energy Measurement Methods

Feature	**Part of Workflow**	**Shell-Script**	**Plugin**	**Prometheus**
No additional software	✓	✓	(✓)	✗
Self-contained	✓	✗	✓	✓
Dealing with workflow faults	✗	✓	✓	✓
Enables full measurement	✗	✓	(✓)	✓
Easy portability	✗	✓	✓	✓
No workflow modifications	✗	✓	(✓)	✓
Low overhead	✓	✓	✓	✗
Multi-tenancy	✗	✗	✗	✗

4.2.1 Manage Energy Measurement as part of the Workflow

In this strategy, the measurement of the RAPL values is managed directly as part of the workflow. The code of the workflow is modified to include additional code that starts and stops the measurement in pods configured with root privileges. That means that in order to utilize this technique for task-based measurement on individual nodes of the cluster, the code of each individual task in the workflow needs to be changed accordingly. Since a direct communication between pods is not possible due to the limitations described in Section 3.3, a workaround is necessary. The workflow is extended with two additional tasks which are executed at the start and at the end of the workflow. The first task writes a file start.txt to a specific location in the file system. The pods for energy measurement continuously check this location in the file system using a daemon that runs in the background. As soon as the file start.txt is detected, they begin the energy measurement by writing the values of the energy counters stored in the RAPL MSRs to a file together with time stamps. As long as the file exists, each pod for energy measurement continues to append the current values of the registers to the file in regular intervals. At the end of the workflow, the second additional task is executed to remove the file start.txt. When that happens, the energy measurement is stopped. The energy values are stored on the shared storage of the cluster, and can

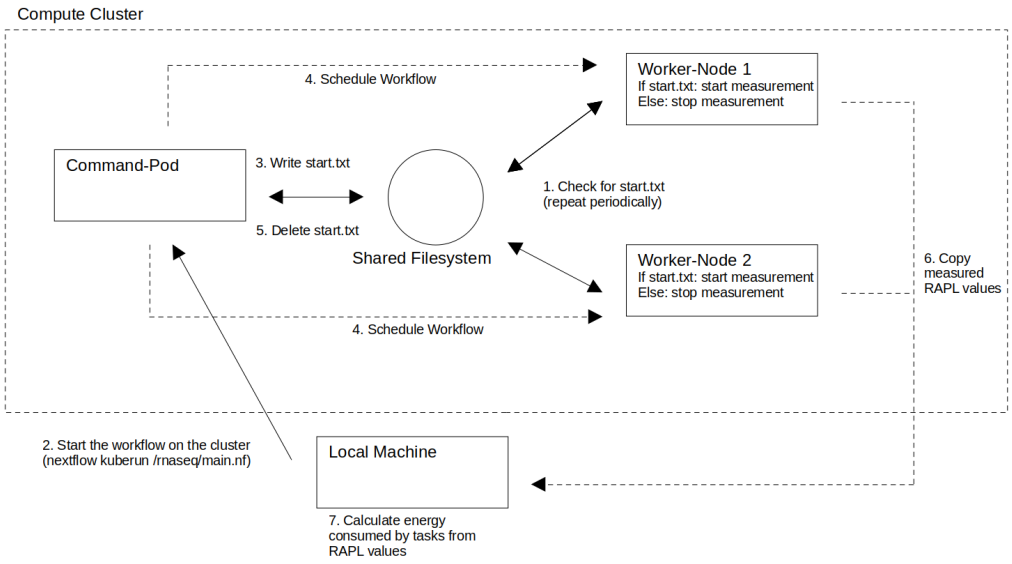

Figure 4: The communication between the machines when controlling the RAPL measurement as part of the workflow. Each arrow represents a communication between two components of the cluster, or between a component of the cluster and the local machine of the user running the workflow. Solid arrows represent communication directly related to the process of measuring RAPL values, while dashed arrows show communication for workflow execution and extraction of results. The numerated annotations describe the actions that are part of the measurement process in ascending order, starting with 1. as the first action and ending with 7. as the last action of the measurement process. Note that the figure shows two worker nodes and one separate command pod. In practice, the number of used worker nodes depends on the cluster configuration and on the executed workflow. The command pod can be executed on a separate node of the cluster or on one of the worker nodes. During the process of energy measurement, it can be treated like any other pod that is part of the executed workflow.

therefore be accessed by any pod. A separate script is then used to calculate the energy consumed by each physical task and copy the results to the local machine of the workflow user. This process enables communication between the pods through the file system and ensures that the energy is measured during the execution of the workflow, but not when the workflow is finished. In order to enable energy measurement for individual physical tasks, the same technique is used to write and delete the file used to start measurements before and after the respective task. Figure 4 shows the communication between the machines during the workflow in order to enable energy measurement.

Advantages and Drawbacks

The method implementing the energy measurement as part of the workflow does not fulfill four of our seven criteria of a good solution for RAPL-based energy measurement. An overview of all criteria for an ideal energy measurement strategy utilizing Intel RAPL and which of them are fulfilled by the individual presented approaches can be found in Table 1. Table 2 shows a general overview of the strengths and weaknesses of each of the presented approaches.

- **No additional software:** No additional software is required for energy measurement. All changes to the software are in the workflow itself.
- **Self-contained:** The approach does not rely on additional external hard- or software, making it self-contained.
- **Dealing with workflow faults:** The measurement process is unable to react to workflow faults. A failing workflow leads to the following tasks not being executed, which also means that the file used as a signal is not deleted automatically and the energy measurement continues. In that case, a manual deletion of the file is necessary in order to stop the energy measurement.
- **Enables full measurement:** The energy measurement is not complete because it is started and stopped by a task of the workflow. Therefore, the energy measurement can only be

started when the workflow is already running and other tasks are already being scheduled, and stopped when it is not finished yet. That means at least the first and last task of the workflow can not be measured completely, leading to slightly incomplete measurements for some tasks of the workflow.

- **Easy portability:** The method is not easily portable to other workflows or even to measure only individual tasks, since additional tasks to write and remove start.txt have to be added to every workflow script individually. If the energy of a new workflow shall be measured, it is necessary to add the code for the measurement.
- **No workflow modifications:** Workflow modifications are necessary to utilize the method, since additional tasks are necessary to create and remove the file start.txt. This increases the workload for the user implementing and using the workflow.
- **Low overhead:** The overhead induced by this method is limited to the additional actions for writing to the file that controls the RAPL monitoring. This amount of additional overhead is negligible. Additionally, two additional tasks for controlling the energy measurement need to be scheduled. However, these tasks are small and executed quickly, making their overhead insignificant.
- **Multi-tenancy:** Since RAPL does not support energy measurements for individual CPU cores, an additional heuristic is necessary to assign a specific amount of energy to a physical task, if multiple tasks are running in parallel on the same CPU. The same is still true if the physical tasks belong to multiple individual workflows running in parallel.

4.2.2 Wrapping Shell-Script

The second method of measuring the energy of a scientific workflow does not change the workflow itself. Instead, the execution of the workflow is wrapped in a shell-script executed on the local machine of the user. This script automatically coordinates the workflow execution with the energy measurement. The shell-script first starts the energy measurement by sending commands to the pods for energy measurement using kubectl. Then the workflow is executed without any changes compared to a normal execution without energy measurement. The script periodically polls if the workflow is still running by checking the status of the coordinating pod of the workflow using kubectl commands. As soon as the workflow finishes running, the script stops the energy measurement and copies the files containing RAPL energy values with timestamps to the local machine of the user for further processing. Figure 5 shows the individual steps of communication between the machines when using this method.

Advantages and Drawbacks

- **No additional software:** Except for the shell-script itself, no additional software is required on the cluster or on the local machine of the workflow user.
- **Self-contained:** A drawback of using a shell-script is that the energy measurement is not self-contained. The local machine of the user needs to run the script during the entire workflow. The machine needs to stay online and connected to the cluster in order to successfully orchestrate the energy measurement.
- **Dealing with workflow faults:** The shell-script can monitor the workflow and the state of the cluster during execution. It is therefore capable of reacting to workflow faults.
- **Enables full measurement:** Since the shell-script controls both the workflow execution and the measurement process, it can start the measurement before initiating the workflow execution. For this reason, full measurement of all workflow tasks is possible.
- **Easy portability:** The shell-script is independent of the executed workflow. To use the script for monitoring a different workflow, only the single line initiating workflow execution must be changed.
- **No workflow modifications:** This method does not require any modifications of the workflow.
- **Low overhead:** This method does not introduce any overhead in the workflow itself. The overhead introduced by the shell-script while communicating with the cluster is negligible.

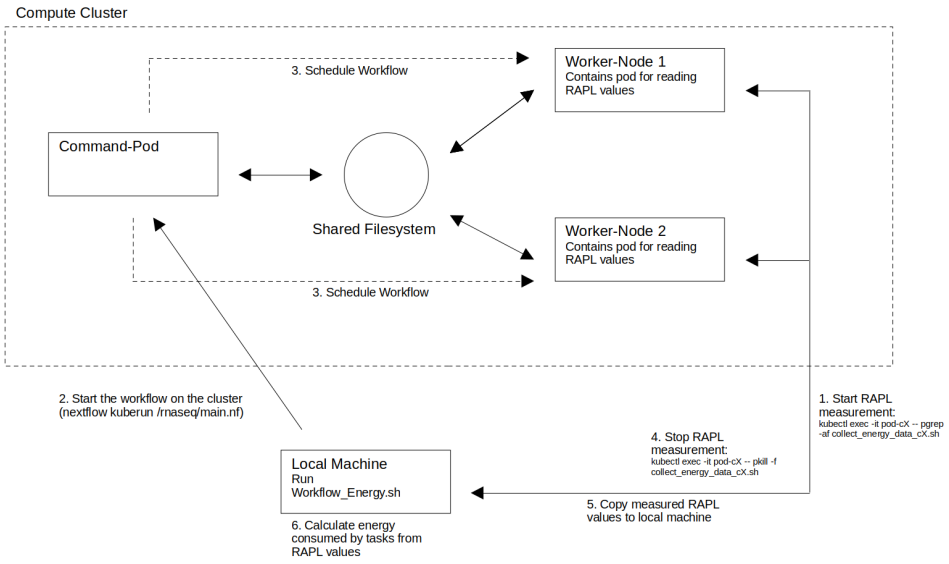

Figure 5: The communication between the machines when controlling the RAPL measurement from the machine of the user using a shell-script. Each arrow represents a communication between two components of the cluster, or between a component of the cluster and the local machine of the user running the workflow. Solid arrows represent communication directly related to the process of measuring RAPL values, while dashed arrows show communication for workflow execution and extraction of results. The numerated annotations describe the actions that are part of the measurement process in ascending order, starting with 1. as the first action and ending with 6. as the last action of the measurement process. Note that the figure shows two worker nodes and one separate command pod. In practice, the number of used worker nodes depends on the cluster configuration and on the executed workflow. The command pod can be executed on a separate node of the cluster or on one of the worker nodes. During the process of energy measurement, it can be treated like any other pod that is part of the executed workflow.

- **Multi-tenancy:** Since RAPL does not support energy measurements for individual CPU cores, an additional heuristic is necessary to assign a specific amount of energy to a physical task, if multiple tasks are running in parallel on the same CPU. The same is still true if the physical tasks belong to multiple individual workflows running in parallel.

4.2.3 Integration through Nextflow-Plugin

A third way of automating the energy measurement through RAPL is by using a Nextflow plugin. Nextflow provides support for plugins that are loaded and executed individually for any workflow. A Nextflow plugin is an extension that enhances Nextflow's functionality by adding custom features or integrations without modifying the core workflow definition. Plugins can hook into different stages of execution, allowing developers to automate specific tasks before, during, or after a workflow runs. A solution utilizing a Nextflow plugin works as shown in Figure 6. This method is similar to the method managing the measurement as part of the workflow (see Section 4.2.1), but no changes are necessary to the workflow definition itself. Instead, the plugin handles writing the files used as a signal to the daemon for energy measurement. This is easier than implementing it as a direct part of the workflow, since Nextflow extensions natively support functions to execute methods before starting the workflow or individual workflow tasks, and after they have stopped running. This also omits the need to schedule the methods as additional tasks of the workflow.

Advantages and Drawbacks

- **No additional software:** The plugin is required as additional software. However, that is barely an issue since the plugin can be integrated using a single command-line parameter, and Nextflow automatically downloads and installs the plugin before executing the workflow, if the plugin has been published in the Nextflow plugins repository[4].

[4]https://github.com/nextflow-io/plugins, last accessed: May 14, 2025

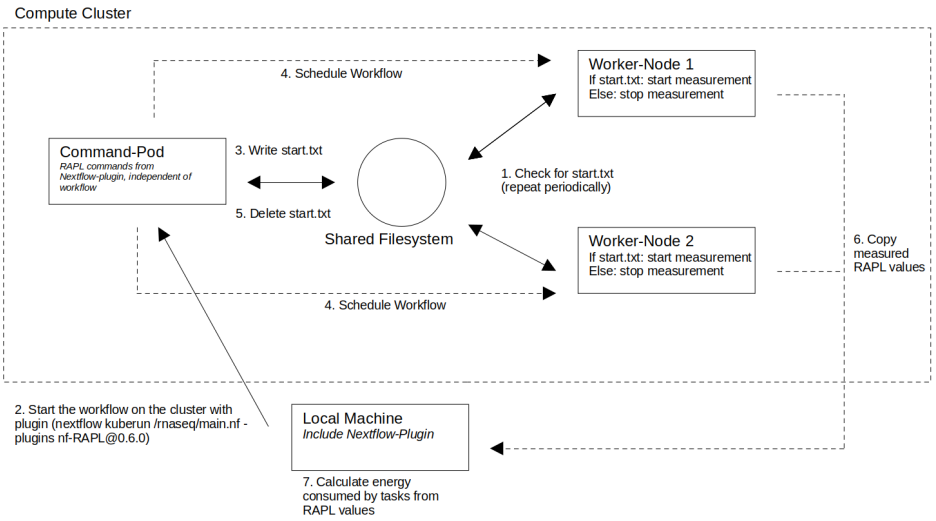

Figure 6: The communication between the machines when controlling the RAPL measurement from inside Nextflow using a workflow-independent plugin that is executed in addition to the workflow. Each arrow represents a communication between two components of the cluster, or between a component of the cluster and the local machine of the user running the workflow. Solid arrows represent communication directly related to the process of measuring RAPL values, while dashed arrows show communication for workflow execution and extraction of results. The numerated annotations describe the actions that are part of the measurement process in ascending order, starting with 1. as the first action and ending with 7. as the last action of the measurement process. Note that the figure shows two worker nodes and one separate command pod. In practice, the number of used worker nodes depends on the cluster configuration and on the executed workflow. The command pod can be executed on a separate node of the cluster or on one of the worker nodes. During the process of energy measurement, it can be treated like any other pod that is part of the executed workflow.

- **Self-contained:** The approach does not rely on additional external hard- or software, making it self-contained. The only exception is the initial download of the plugin from an external source. However, the plugin can also be installed manually, removing this dependency.
- **Dealing with workflow faults:** The plugin can actively react to the current state of the workflow and take appropriate action if a fault occurs.
- **Enables full measurement:** The implementation through a Nextflow plugin technically does not allow measuring the energy consumption of the whole workflow execution, because the workflow is already running when the measurement is started. But the plugin allows starting the measurement before the first task is executed and to end it after the last task has stopped running, enabling full measurement of every single task of the workflow.
- **Easy portability:** Since the plugin can be integrated using a single line of code, integrating it into different workflows is trivial.
- **No workflow modifications:** Technically, the workflow needs to be modified to include the plugin. But since this modification consists of either a single line of code or a single command line parameter, the effort is negligible.
- **Low overhead:**The plugin only takes action at the beginning and end of workflow tasks, making the overhead negligible.
- **Multi-tenancy:** Since RAPL does not support energy measurements for individual CPU cores, an additional heuristic is necessary to assign a specific amount of energy to a physical task, if multiple tasks are running in parallel on the same CPU. The same is still true if the physical tasks belong to multiple individual workflows running in parallel.

4.2.4 Using Prometheus

Prometheus[5] is an open-source monitoring solution that automatically records time-series data of various metrics of a system and allows the user to access this data through queries. Prometheus can be deployed on each node of a compute cluster and supports reading data from Intel RAPL registers with additional configuration. By default, it can read a different set of energy values collected at the power supply. This data includes the energy consumed by the whole system. It is read through the Intelligent Platform Management Interface (IPMI) utilizing the Advanced Configuration and Power Interface (ACPI). That makes using Prometheus a fourth method to collect energy data by letting Prometheus periodically collect the data from the registers while a workflow is running. After the workflow has finished, the collected data can be accessed by connecting to the Prometheus UI and writing a query to get the energy consumption over a specific period of time on a specific node, e.g., the duration and end time of a task that was executed as part of a workflow on the node. A query for Kubernetes can look as follows:

```
sum_over_time(node_hwmon_power_average_watt{instance=~"10.0.0.37:9100|
10.0.0.38:9100"}[1476s] @ 1743495765) * 30
```

This query returns the cumulative consumed energy in Joule of the nodes with the IP-addresses 10.0.0.37 and 10.0.0.38 over a period of 1476 seconds (the duration of the workflow) at the time 10:22:45 formatted as a Unix timestamp (1743495765) for a scraping interval of 30 seconds.

Advantages and Drawbacks

- **No additional software:** In order to us Prometheus to read data regarding energy consumption from a cluster, it is necessary to install additional software on all nodes of the cluster. While this is sometimes unproblematic, it can be unwanted or even impossible in other cases, making the method utilizing Prometheus not universally applicable.
- **Self-contained:** Prometheus is self-contained. It runs on every node, independent of any other software. The information regarding energy consumption can be accessed directly through its own interface.
- **Dealing with workflow faults:** Since Prometheus is completely independent of the workflow, values from the RAPL registers are collected even if the workflow fails.
- **Enables full measurement:** RAPL counters are monitored continuously by Prometheus. Therefore, it is always possible to query for the period of time when the workflow or any of its tasks were executed.
- **Easy portability:** Prometheus works with any workflow if it is executed on the monitored cluster.
- **No workflow modifications:** Prometheus is completely independent of the software running on the cluster. For this reason, no modifications of the workflow are necessary.
- **Low overhead:** Unlike the other presented methods, Prometheus runs at all times on all nodes and continuously collects data regarding energy consumption and other metrics. This adds a significant computational overhead to the cluster, especially for high workloads [9], thereby reducing the energy efficiency of the cluster.
- **Multi-tenancy:** Since RAPL does not support energy measurements for individual CPU cores, an additional heuristic is necessary to assign a specific amount of energy to a physical task, if multiple tasks are running in parallel on the same CPU. The same is still true if the physical tasks belong to multiple individual workflows running in parallel.

5 Experiments

We implemented all methods and tested them in combination with scientific workflows from three different domains of research (Genomics, Proteomics and Remote Sensing) to study the amount of energy measured by each method while executing the same workflow. We then compare the amount of measured energy between the approaches for singular tasks and the whole workflow. This enables us to draw conclusions about the viability of each method to coordinate the energy measurement and capture accurate values across all tasks of a workflow.

[5] https://prometheus.io/, last accessed: May 14, 2025

Table 2: Strengths and Weaknesses of Energy Measurement Methods

Method	Strengths (✓)	Weaknesses (✗)
As part of the Workflow	No additional software Self-contained on cluster	Not all tasks measured Code integration per workflow
Wrapping Shell-Script	No modifications required Full measurement possible	Manual adjustments needed Local machine communication
Nextflow Plugin	Easy CLI integration Fully measures workflow	Command pod not measured fully
Prometheus	Continuous measurement No workflow changes needed	Requires extra software Increased resource consumption

5.1 Hardware Setup for Implementation

For our implementations and testing, we used a commodity cluster consisting of 14 nodes in total, of which two were available for our experiments with exclusive access. Each node of the cluster contains an Intel(R) Xeon(R) Silver 4314 CPU running at a frequency of 2.40GHz, 256GB of main memory and 8TB of mass storage. The CPUs in the cluster support the Package and DRAM domains of Intel RAPL. The cluster uses Kubernetes for orchestration, which in turn deploys Docker containers. Our workflows were programmed in Groovy using Nextflow. We deployed them to the cluster utilizing the Nextflow interface for Kubernetes (nextflow kuberun) or by executing them directly in a pod on the cluster (nextflow run), depending on the method.

5.2 Used Workflows

To conduct our experiments, we used a set of three different scientific workflows obtained from nf-core[6]. The workflows are all implemented in Nextflow and can be executed on a compute cluster using Kubernetes. They feature a broad range of tasks and are used in different scientific domains. Therefore, we consider them as well suited for experiments to examine the performance of the proposed methods for energy measurement. Table 3 shows an overview of the three workflows used in our experiments.

RNASeq

RNASeq[7] is a scientific workflow from the area of bioinformatics. It is used to analyze the RNA sequencing data obtained from organisms with a reference genome and annotation. RNASeq contains only few tasks, but some of them are very compute and memory intensive, causing long runtimes. We use a modified version of this workflow with a simplified pipeline for our tests. This pipeline only contains the tasks *fastp*, *star_index*, *fastqsplit*, *star_align*, *samtools*, *samtools_merge* and *cufflinks*. All other components contained in the nf-core version of RNASeq were removed for simplified testing.

Quantms

Quantms[8] is a bioinformatics workflow from the area of Proteomics. Among other tasks, it can be used for label-free quantification of Quantitative Mass Spectrometry data. Quantms contains a larger set of tasks than RNASeq, of which most are very small and less resource-intensive than those in RNASeq. It therefore provides a good contrast to RNASeq for our experiments.

Rangeland

Rangeland[9] is an analysis pipeline from the area of Remote Sensing in Geography. It is used to process satellite imagery in order to assess changes in land-cover over time. During execution, the scientific workflow processes large sets of tasks in parallel, each processing one of the images in the dataset. This provides us with information about the influence of numerous small tasks, which are executed rapidly and in quick succession, on our measurement strategies.

[6]https://nf-co.re/pipelines/, last accessed: May 14, 2025
[7]https://nf-co.re/rnaseq/3.18.0/, last accessed: May 14, 2025
[8]https://nf-co.re/quantms/1.2.0/, last accessed: May 14, 2025
[9]https://nf-co.re/rangeland/1.0.0/, last accessed: May 14, 2025

Table 3: Overview of the scientific workflows used in our experiments

Workflow	Domain	No. phys. Tasks	Input Size	Output Size	Runtime
RNASeq	Bioinformatics	9	5.7 GB	390 MB	25m
Quantms	Proteomics	61	100 MB	47 MB	2m 10s
Rangeland	Remote Sensing	280	250 MB	350 MB	3m 10s

5.3 Experimental Strategy

To compare the amount of energy measured during workflow execution by each of the four methods, we run each of the three described scientific workflows on a compute cluster. In order to keep the runtime for each test manageable (below 30 minutes per run for each workflow) and the tests realistic, we use small input datasets of real-world data. Since the four methods for energy measurement all read the same RAPL energy counters (except Prometheus), it is not necessary to run individual experiments for each method separately. Instead, we evaluate the methods simultaneously by running each workflow and collecting the values of the RAPL energy counters during execution using each of the proposed methods. This has the advantage that we save time and energy. To calculate the amount of energy captured by each of the methods (shell-script, plugin and task-based), we determine the points in time where each of the methods starts and stops the measurement. We then use the same collected RAPL values for each method to extract the exact amount of captured energy consumed by the workflow. This experiment is repeated five times for each workflow, and the averages of all runs are used during our evaluation. Note that the experiments for the method utilizing Prometheus are run separately to ensure that all data from Prometheus can be collected shortly after the experiment and using the correct scraping interval.

5.4 Results

Complete Workflow

Our experimental results regarding the absolute and relative differences in measured energy consumption between the three approaches are shown in Table 4. The table shows that only the method based on a shell-script is able to capture the full energy consumption of the different workflows. The other two methods are missing some of the energy included in the RAPL counter values. The plugin-based and task-based methods miss about 0.19% (754J) and 0.36% (1437J) compared to the shell-script on RNASeq. On Quantms and Rangeland, the absolute losses are higher, with about 7.08% (2053J) and 5.22% (1852J) for the plugin and 7.67% (2223J) and 5.51% (1952J) for the task-based method, respectively. This is due to the fact that Quantms and Rangeland load additional plugins before starting the energy measurement for both the plugin-based and task-based method, delaying the start of the measurement by a few seconds (from between 3.6s and 6.2s on average for RNASeq to 9.8s and 10.8s for Quantms and 9.2s and 9.8s for Rangeland).

The higher delays causing larger differences in measured energy consumption are also part of the cause for the higher relative differences between the shell-script and the methods based on a plugin or task-based management. However, they do not explain the differences between the three workflows alone. The largest factor for the small relative difference between the approaches of only 0.36% for RNASeq compared to 7.67% for Quantms and 5.51% for Rangeland is their runtime. RNASeq runs considerably longer (about 25 min) on our test dataset than Quantms (about 2 min) or Rangeland (about 3 min). Since most of the energy consumption of the tested workflows is caused by computationally intensive tasks in the middle of the workflow and not right at the start, the energy missed at the beginning of the workflow by some of the tested methods is amortized quickly, if the workflow has a sufficiently long runtime.

Table 5 shows the energy values extracted using Prometheus with a measurement interval of 30 seconds in comparison to those calculated with a shell-script. The amount of energy measured by Prometheus for the longer RNASeq workflow is slightly lower. This tendency is consistent across all test runs. When examining the two workflows with shorter runtimes, the difference between the values returned by Prometheus and the shell-script becomes much larger, with over 30% and 36% respectively for Quantms and Rangeland.

To examine how these results are affected by the measurement interval of Prometheus, we ran the same experiments with a configured measurement interval of 10 seconds. The results of these experiments are shown in Table 6. Changing the measurement interval of Prometheus to 10 seconds does not reduce the difference to the energy measured using a shell-script for all workflows.

Table 4: Energy Consumption: Absolute (J) and Relative (%) per Workflow. Note the much lower energy consumption of Quantms and Rangeland due to the short runtime caused by very small inputs.

Workflow	**Abs Energy (J)**			**Rel Energy (%)**		
	RNASeq	**Quantms**	**Rangeland**	**RNASeq**	**Quantms**	**Rangeland**
Shell-script	393 906.17	28 981.12	35 444.69	100.00	100.00	100.00
Plugin	393 151.76	26 928.59	33 592.85	99.81	92.92	94.78
Task	392 469.08	26 758.27	33 492.20	99.64	92.33	94.49

Table 5: Comparison of measured energy consumption using Prometheus with a measurement interval of 30 seconds and a Shell-script. Note the much lower energy consumption of Quantms and Rangeland due to the short runtime caused by very small inputs.

Dataset	**Prometheus**	**Shell-script**	**Difference**
RNASeq	380 812.00J	395 829.96J	−3.94%
Quantms	44 346.00J	31 015.71J	30.06%
Rangeland	54 432.00J	34 666.70J	36.31%

While the gap between measurements becomes smaller for Quantms and Rangeland, it increases for RNASeq. This shows that the differences in energy consumption measured using Prometheus in comparison to the other methods can not be attributed to the measurement interval alone.

Table 6: Comparison of measured energy consumption using Prometheus with a measurement interval of 10 seconds and 30 seconds.

Dataset	**Shell-script**	**Prometheus (10s)**	**Prometheus (30s)**
RNASeq	396 721.28J	376 000.00J	380 812.00J
Quantms	30 115.31J	42 486.67J	44 346.00J
Rangeland	37 508.91J	49 860.00J	54 432.00J

For all three workflows, a shorter measurement interval leads to a reduction in measured energy consumption. This seems counterintuitive, since Prometheus saves the average energy consumption during the measurement interval at the end of each interval. The total energy consumption is then calculated by adding all saved energy values whose time stamps are during the period of measurement. Since any additional energy consumption included at the start of the measured period due to the first included interval being partly outside of the measured period should be offset at the end of the measured period, a change in the measurement interval should not lead to a consistent reduction in measured energy consumption. We hypothesize that there might be two possible reasons for the differences. The first possible reason for the difference is the method to calculate the average energy consumption during each measurement interval. At the time of writing, it is not entirely clear to the authors how these averages are calculated. It is possible that shorter measurement intervals lead to systematically smaller calculated averages for each interval. A second possibility are changes in the computational load of the machines between experiments. Since Prometheus can only collect values with one measurement interval at a time, our experiments had to be conducted one after the other, with changes to the configuration of Prometheus in between. Although we aimed to guarantee equal conditions during the tests, these changes or other external factors might have lead to a reduced base-load on the machines under test during the second part of our experiments, leading to lower energy consumption being recorded for the shorter measurement interval.

Task-based

Utilizing the presented methods for RAPL-based energy measurement, task-based measurement is only possible for sufficiently long, non-overlapping tasks without introducing errors or resorting to heuristics. For workflows like RNASeq, where each task runs for longer than one second and each node only executes a single task at a time, the energy consumption of each task can be calculated by calculating the energy consumption on its respective node between the start time and the end

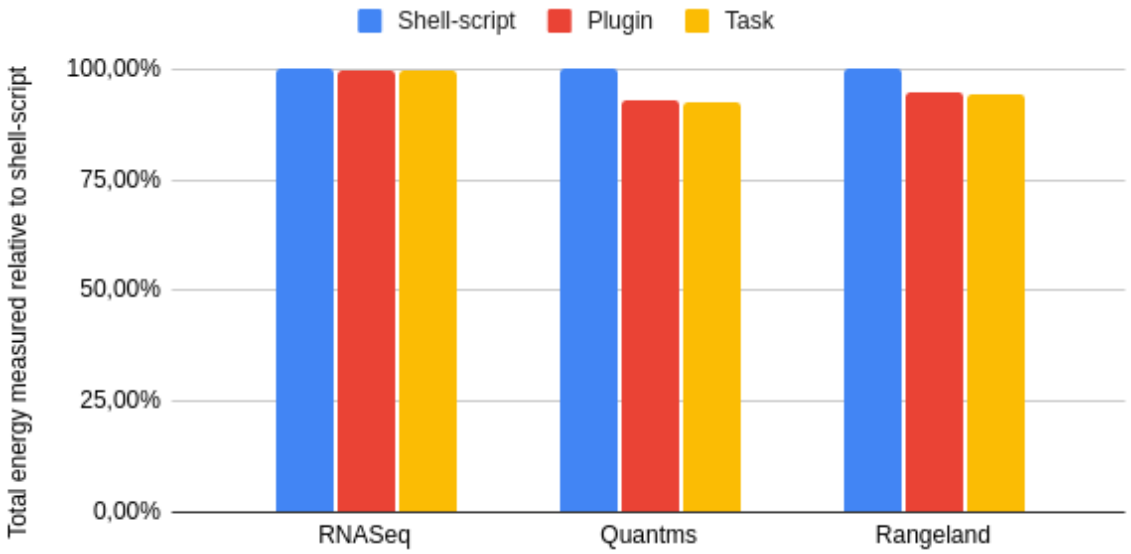

Figure 7: The bar chart shows the differences in the total amount of energy measured using a plugin or an additional workflow task in relation to the amount measured using a shell-script. All values are provided as a percentage of the energy measured with a shell-script, since this method allows starting the measurement before the workflow is initialized. This enables capturing the full energy consumption of the workflow that is measurable with RAPL.

time of the task. Adding up the energy consumption of all tasks results in the energy consumption for the whole workflow.

If the workflow runs tasks concurrently, their exact energy consumption can not be measured individually. Instead, it is only possible to approximate the energy consumption of a task by using heuristics to estimate how much of the total energy consumption of the node in that time window was caused by this particular task. Very short tasks like the computations on individual pictures appearing in Rangeland can cause a similar issue. If the total runtime of the task is below one second, Nextflow logs the same value for the start- and end-time of the task. In that case, it is not possible to calculate the energy consumption based on the exact runtime, but only on an estimated runtime below one second.

6 Discussion

Accuracy of Methods

In Section 4 we present four ways to measure the energy consumption of the system by reading the RAPL energy counters during workflow execution on a compute cluster orchestrated by Kubernetes. Our experiments show that the measured energy consumptions of the methods using a shell-script, a plugin and task-based management are very similar. As presented in Figure 7, even in an unfavorable situation with a very short workflow (like Quantms in our experiments), the task-based method captures more than 92% of the energy used by the workflow. For workflows and datasets with longer runtime like our version of RNASeq, the difference in measured energy is only 0.36%. We expect this number to become even smaller for workflows with longer runtimes, which often appear in real settings of workflow usage, since the difference is entirely caused by the delay before starting the energy measurement for some of the presented methods. This makes all three methods viable candidates for accurate energy measurement for longer workflows. However, if certainty is required that the energy measurement captures 100% of the energy consumption that is included in RAPL energy counters during the whole workflow execution, only the method based on a shell-script can guarantee full coverage.

Figure 8 shows the energy consumption reported by Prometheus in comparison to that measured by the method using a shell-script, the most accurate of the proposed methods. We configured a polling interval of 30 seconds in Prometheus for our experiments. The results show that the energy consumption reported by Prometheus closely matches that of the shell-script for longer workflows such as RNASeq. It is consistently slightly lower across all runs. For short workflows such as Quantms and Rangeland, however, the energy consumption reported by Prometheus is significantly

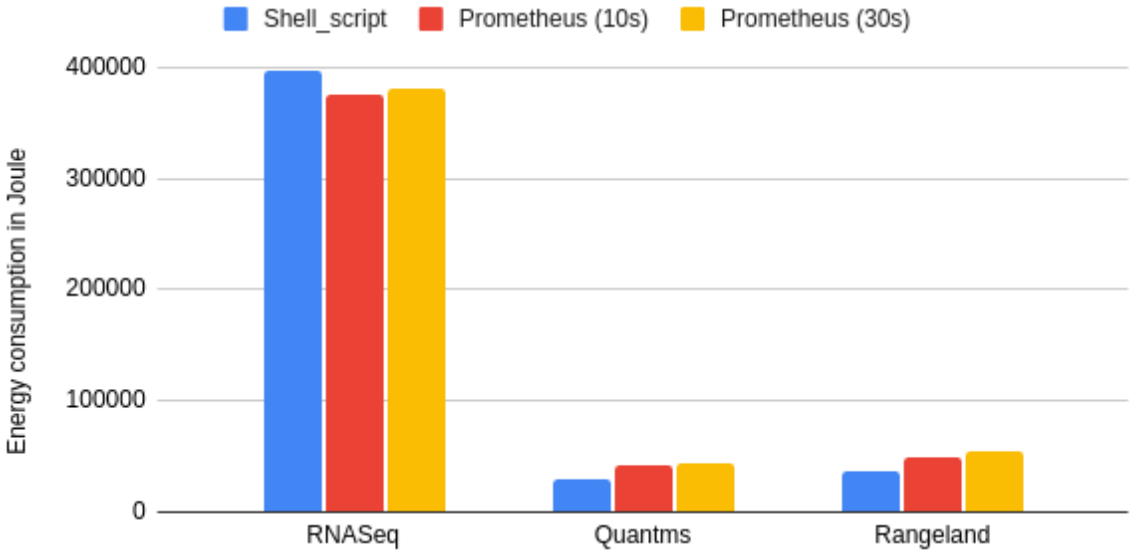

Figure 8: The bar chart shows the differences in the total amount of energy measured using Prometheus with a measurement interval of 10 seconds and 30 seconds, compared to the amount measured using a shell-script.

higher than that measured with the shell-script. For both workflows, there were multiple runs where the reported energy consumption was more than 40% higher than that measured by the shell-script. We hypothesize that one contributing factor to the increased reported energy consumption is the long polling interval of 30 seconds. Since each data point reported by Prometheus represents the energy consumption of the previous 30 seconds and all data points collected during workflow execution are included when calculating the energy consumption of the workflow, some energy might be included in the data points that was consumed outside the time of workflow execution.

To test if this energy significantly impacts results, we ran an additional set of experiments with a polling interval of 10 seconds configured in Prometheus. The energy measured in these experiments (shown in Figure 8) is slightly lower across all workflows. This supports the hypothesis that the long polling interval of 30 seconds does affect the measured energy. However, the differences in measured energy between Prometheus and the shell-script do not decrease by much. In fact, the gap becomes larger for RNASeq. Therefore, the different results between the shell-script and Prometheus are not caused by the measurement interval, but by other differences in the measurement. Finding the exact nature of these differences remains future work.

Initial tests for the measurement of long-term and idle energy consumption show a similar pattern. The energy measured by Prometheus with a polling interval of ten seconds during idle over a period of one hour is around 36.5% higher than that measured with RAPL. These results are consistent over multiple runs. Since we assume that Prometheus uses the same RAPL energy counters for energy measurement, the results suggest that Prometheus either uses some additional metric to include the energy consumption of other components in its output, or some unknown factor leads to higher estimations for the used energy in scenarios with low loads on the CPU and RAM. These findings underscore the need for a deeper investigation into Prometheus's energy reporting methods, in order to ensure that the reported values are interpreted correctly.

Complexity of Implementation

As a measure of the complexity of implementing the three methods not using dedicated cluster monitoring software, we counted the lines of code (LoC) required to implement each of them. In these numbers, we do not include the Python program to calculate the consumed energy from the RAPL energy values and the information about the workflow execution collected by Nextflow. This Python program has 381 LoC in total. Additionally, the bash-scripts used to read RAPL energy values with 24 LoC are not included either.

To implement task-based management, the amount of code required depends on the specific goal. It does require only 20 LoC in the workflow to measure from the start to the end of the workflow execution. However, it should be noted that the complexity of this implementation increases if singular tasks should be measured, due to the necessity to implement the starting and

stopping procedure for each individual task. That means that task-based management requires 20 LoC for each individual task to be measured. This can result in high complexity of implementation if the workflow contains a large number of individual tasks. Also, this method must be implemented in each workflow individually.

The highest number of total lines of code are required to implement a Nextflow plugin. This method requires 105 LoC (excluding comments and empty lines) distributed over multiple files. While the structure of the Nextflow plugin is more complex than an implementation directly inside the workflow, it should be noted that most of the code can be adapted from existing plugins. This reduces the complexity of implementing a Nextflow plugin. If only the lines that need to be changed from existing plugins by the developer are counted, a Nextflow plugin is easier to implement than task-based management. In addition, a plugin has the advantage that it can be ported to a different workflow with only a single line of code in the workflow itself and no changes to the plugin. This makes a plugin the easier to implement choice if used across multiple workflows.

The method utilizing a shell-script contains 64 LoC in total. This is more than the other two methods require at the minimum, but the script contains additional code to start and monitor the workflow, delete old data, copy the files produced by the workflow and the energy measurement to the local machine and start the Python program to convert the RAPL values to meaningful energy values. This makes the script more laborious to implement initially, but it can save a lot of time in the long run by automating processes. Furthermore, it can be adapted to new workflows by changing only a single line of code.

7 Future Work

In this section, we present directions for future work in order to further improve the energy measurements using RAPL and to measure and improve the energy efficiency of workflows executed on commodity clusters in general.

7.1 Validate Accuracy of RAPL for Energy Measurement

One area of future work is to validate the accuracy of energy measurements using RAPL in the context of compute clusters. While the general accuracy of RAPL has been validated [15], it is currently unclear if RAPL provides a complete picture of the energy consumed by scientific workloads executed on compute clusters, where aspects such as the energy used by storage or the network between nodes may have a larger impact than in other environments. To validate the accuracy of RAPL, experiments comparing the energy usage measured by RAPL and hardware energy measurement tools [16], while executing scientific workflows on compute clusters, could be conducted. While the accuracy of hardware energy measurement tools [16], RAPL [7, 15] and software models [23] has been compared in other contexts [6], there have been no such experiments in the context of workflows and cluster computing. Due to the fact that some scientific workflows are characterized by huge datasets that require much more loading of data from memory and storage as many other workloads, the absolute and relative accuracy of the methods for energy measurement might be different for these computations.

7.2 Implement Concurrent Task-level Energy Measurement

While our implementation is capable of measuring the energy consumption of whole workflows and single tasks that run isolated from other tasks, additional work is necessary to approximate the energy consumption of tasks that run concurrently on the same node. Since RAPL counters in the Package domain can only measure the energy of the whole CPU, it is not possible to measure these tasks separately. Instead, a heuristic is necessary to estimate for how much of the consumed energy each of the tasks is responsible. Such a metric could be based on different statistics [23], e.g. CPU time, utilized DRAM or the amount of CPU cores used.

7.3 Predict Energy Consumption of Workflows

When methods to measure the energy consumption of workflows executed on commodity clusters have been developed and proven to be reliable, the results of these measurements can be used together with metadata of the workflow executions in order to predict the energy consumption of future workflow executions based on the configuration of the workflow and the cluster. If reliable predictions of the energy consumption of a workflow are possible, they could be used to optimize the energy consumption of a workflow execution without having to execute the workflow beforehand.

If runtime predictions are also possible, a two-dimensional optimization space is created that can be used to find the optimal trade-off between runtime and energy consumption. Similar work has already been published for cloud environments [22] and utilizing dynamic voltage and frequency scaling (DVFS) [29].

8 Conclusion

In this work, we present four ways to read the RAPL energy counters and IPMI energy values on a compute cluster orchestrated by Kubernetes while executing a scientific workflow, in order to calculate the energy consumed by the cluster's CPU and DRAM during the execution. We discuss the workings, advantages and drawbacks of each of these approaches, experimentally show that they produce similar results and compare their complexity. We conclude that utilizing a Nextflow plugin or Prometheus offer the most benefits, if the necessary software is available. In cases where no additional software can be used, a shell-script provides a better experience to the workflow developer than implementing the energy measurement as part of the workflow. We also provide an overview of the Intel RAPL feature and highlight topics and ideas to be investigated in the future. Overall, this work helps the reader get an overview of the pitfalls of energy measurement, especially in the context of compute clusters orchestrated by Kubernetes, and provides a starting point to implement energy monitoring for workflow optimization in similar contexts.

References

[1] Lukas Alt, Anara Kozhokanova, Thomas Ilsche, Christian Terboven, and Matthias S. Mueller. An Experimental Setup to Evaluate RAPL Energy Counters for Heterogeneous Memory. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering*, pages 71–82, London United Kingdom, May 2024. ACM.

[2] Jonathan Bader, Fabian Lehmann, Lauritz Thamsen, Ulf Leser, and Odej Kao. Lotaru: Locally predicting workflow task runtimes for resource management on heterogeneous infrastructures. *Future Generation Computer Systems*, 150:171–185, January 2024.

[3] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. *The Datacenter as a Computer*. Springer US, third edition edition, 2018.

[4] Daniel Bedard, Robert Fowler, Min Yeol Lim, and Allan Porterfield. PowerMon 2: Fine-grained, Integrated Power Measurement.

[5] Carmen Carrión. Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges. *ACM Computing Surveys*, 55(7):1–37, July 2023.

[6] Fernando Castor. Estimating the Energy Footprint of Software Systems: a Primer, July 2024. arXiv:2407.11611 [cs].

[7] Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. RAPL: memory power estimation and capping. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, pages 189–194, Austin Texas USA, August 2010. ACM.

[8] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35(4):316–319, April 2017.

[9] Madalina Dinga, Ivano Malavolta, Luca Giamattei, Antonio Guerriero, and Roberto Pietrantuono. An Empirical Evaluation of the Energy and Performance Overhead of Monitoring Tools on Docker-Based Systems. In Flavia Monti, Stefanie Rinderle-Ma, Antonio Ruiz Cortés, Zibin Zheng, and Massimo Mecella, editors, *Service-Oriented Computing*, volume 14419, pages 181–196. Springer Nature Switzerland, Cham, 2023. Series Title: Lecture Notes in Computer Science.

[10] Juan J. Durillo, Hamid Mohammadi Fard, and Radu Prodan. MOHEFT: A multi-objective list-based method for workflow scheduling. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 185–192, Taipei, Taiwan, December 2012. IEEE.

[11] James A. Fellows Yates, Thiseas C. Lamnidis, Maxime Borry, Aida Andrades Valtueña, Zandra Fagernäs, Stephen Clayton, Maxime U. Garcia, Judith Neukamm, and Alexander Peltzer. Reproducible, portable, and efficient ancient genome reconstruction with nf-core/eager. *PeerJ*, 9:e10947, March 2021.

[12] Charlotte Freitag, Mike Berners-Lee, Kelly Widdicks, Bran Knowles, Gordon S. Blair, and Adrian Friday. The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations. *Patterns*, 2(9):100340, September 2021.

[13] Yansong Gao, Huming Qiu, Zhi Zhang, Binghui Wang, Hua Ma, Alsharif Abuadbba, Minhui Xue, Anmin Fu, and Surya Nepal. DeepTheft: Stealing DNN Model Architectures through Power Side Channel, September 2023. arXiv:2309.11894 [cs].

[14] Yolanda Gil, Ewa Deelman, Mark Ellisman, Thomas Fahringer, Geoffrey Fox, Dennis Gannon, Carole Goble, Miron Livny, Luc Moreau, and Jim Myers. Examining the Challenges of Scientific Workflows. *Computer*, 40(12):24–32, December 2007.

[15] Daniel Hackenberg, Robert Schone, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. An Energy Efficiency Feature Survey of the Intel Haswell Processor. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 896–904, Hyderabad, India, May 2015. IEEE.

[16] Mathilde Jay, Vladimir Ostapenco, Laurent Lefevre, Denis Trystram, Anne-Cécile Orgerie, and Benjamin Fichel. An experimental comparison of software-based power meters: focus on CPU and GPU. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 106–118, Bangalore, India, May 2023. IEEE.

[17] Richard Kavanagh and Karim Djemame. Rapid and accurate energy models through calibration with IPMI and RAPL. *Concurrency and Computation: Practice and Experience*, 31(13):e5124, July 2019.

[18] Hamidreza Khaleghzadeh, Muhammad Fahad, Arsalan Shahid, Ravi Reddy Manumachu, and Alexey Lastovetsky. Bi-Objective Optimization of Data-Parallel Applications on Heterogeneous HPC Platforms for Performance and Energy Through Workload Distribution. *IEEE Transactions on Parallel and Distributed Systems*, 32(3):543–560, March 2021.

[19] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K. Nurminen, and Zhonghong Ou. RAPL in Action: Experiences in Using RAPL for Power Measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 3(2):1–26, June 2018.

[20] James H. Laros, Phil Pokorny, and David DeBonis. PowerInsight - A commodity power measurement capability. In *2013 International Green Computing Conference Proceedings*, pages 1–6, Arlington, VA, USA, June 2013. IEEE.

[21] Fabian Lehmann, Jonathan Bader, Ninon De Mecquenem, Xing Wang, Vasilis Bountris, Florian Friederici, Ulf Leser, and Lauritz Thamsen. Ponder: Online Prediction of Task Memory Requirements for Scientific Workflows. In *2024 IEEE 20th International Conference on e-Science (e-Science)*, pages 1–10, September 2024. arXiv:2408.00047 [cs].

[22] Zhongjin Li, Jidong Ge, Haiyang Hu, Wei Song, Hao Hu, and Bin Luo. Cost and Energy Aware Scheduling Algorithm for Scientific Workflows with Deadline Constraint in Clouds. *IEEE Transactions on Services Computing*, 11(4):713–726, July 2018.

[23] Weiwei Lin, Fang Shi, Wentai Wu, Keqin Li, Guangxin Wu, and Al-Alas Mohammed. A Taxonomy and Survey of Power Models and Power Modeling for Cloud Servers. *ACM Computing Surveys*, 53(5):1–41, September 2021.

[24] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 355–371, San Francisco, CA, USA, May 2021. IEEE.

[25] Bertram Ludäscher, Mathias Weske, Timothy McPhillips, and Shawn Bowers. Scientific Workflows: Business as Usual? In Umeshwar Dayal, Johann Eder, Jana Koehler, and Hajo A. Reijers, editors, *Business Process Management*, volume 5701, pages 31–47. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. Series Title: Lecture Notes in Computer Science.

[26] Felix Mölder, Kim Philipp Jablonski, Brice Letcher, Michael B. Hall, Christopher H. Tomkins-Tinch, Vanessa Sochat, Jan Forster, Soohyun Lee, Sven O. Twardziok, Alexander Kanitz, Andreas Wilm, Manuel Holtgrewe, Sven Rahmann, Sven Nahnsen, and Johannes Köster. Sustainable data analysis with Snakemake. *F1000Research*, 10:33, April 2021.

[27] Kenneth O'brien, Ilia Pietri, Ravi Reddy, Alexey Lastovetsky, and Rizos Sakellariou. A Survey of Power and Energy Predictive Models in HPC Systems and Applications. *ACM Computing Surveys*, 50(3):1–38, May 2018.

[28] James Phung, Young Choon Lee, and Albert Y. Zomaya. Modeling System-Level Power Consumption Profiles Using RAPL. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–4, Cambridge, MA, November 2018. IEEE.

[29] Ilia Pietri and Rizos Sakellariou. Energy-Aware Workflow Scheduling Using Frequency Scaling. In *2014 43rd International Conference on Parallel Processing Workshops*, pages 104–113, Minneapolis, MN, USA, September 2014. IEEE.

[30] Babak Bashari Rad, Harrison John Bhatti, and Mohammad Ahmadi. An Introduction to Docker and Analysis of its Performance. 2017.

[31] Robert Schöne, Thomas Ilsche, Mario Bielert, Markus Velten, Markus Schmidl, and Daniel Hackenberg. Energy Efficiency Aspects of the AMD Zen 2 Architecture. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 562–571, September 2021. arXiv:2108.00808 [cs].

[32] Satyabrata Sen, Neena Imam, and Chung-Hsing Hsu. Quality Assessment of GPU Power Profiling Mechanisms. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 702–711, Vancouver, BC, May 2018. IEEE.

[33] Khaldoun Senjab, Sohail Abbas, Naveed Ahmed, and Atta Ur Rehman Khan. A survey of Kubernetes scheduling algorithms. *Journal of Cloud Computing*, 12(1):87, June 2023.

[34] Viktória Spišaková, Lukáš Hejtmánek, and Jakub Hynšt. Nextflow in Bioinformatics: Executors Performance Comparison Using Genomics Data. *Future Generation Computer Systems*, 142:328–339, May 2023.

[35] Martin Sudmanns, Dirk Tiede, Hannah Augustin, and Stefan Lang. Assessing global Sentinel-2 coverage dynamics and data availability for operational Earth observation (EO) applications using the EO-Compass. *International Journal of Digital Earth*, 13(7):768–784, July 2020.

[36] The Galaxy Community, Enis Afgan, Anton Nekrutenko, Bjórn A Grüning, Daniel Blankenberg, Jeremy Goecks, Michael C Schatz, Alexander E Ostrovsky, Alexandru Mahmoud, Andrew J Lonie, Anna Syme, Anne Fouilloux, Anthony Bretaudeau, Anton Nekrutenko, Anup Kumar, Arthur C Eschenlauer, Assunta D DeSanto, Aysam Guerler, Beatriz Serrano-Solano, Bérénice Batut, Björn A Grüning, Bradley W Langhorst, Bridget Carr, Bryan A Raubenolt, Cameron J Hyde, Catherine J Bromhead, Christopher B Barnett, Coline Royaux, Cristóbal Gallardo, Daniel Blankenberg, Daniel J Fornika, Dannon Baker, Dave Bouvier, Dave Clements, David A De Lima Morais, David Lopez Tabernero, Delphine Lariviere, Engy Nasr, Enis Afgan, Federico Zambelli, Florian Heyl, Fotis Psomopoulos, Frederik Coppens, Gareth R Price, Gianmauro Cuccuru, Gildas Le Corguillé, Greg Von Kuster, Gulsum Gudukbay Akbulut, Helena Rasche, Hans-Rudolf Hotz, Ignacio Eguinoa, Igor Makunin, Isuru J Ranawaka, James P Taylor, Jayadev Joshi, Jennifer Hillman-Jackson, Jeremy Goecks, John M Chilton, Kaivan Kamali, Keith Suderman, Krzysztof Poterlowicz, Le Bras Yvan, Lucille Lopez-Delisle, Luke Sargent, Madeline E Bassetti, Marco Antonio Tangaro, Marius Van Den Beek, Martin Čech, Matthias Bernt, Matthias Fahrner, Mehmet Tekman, Melanie C Föll, Michael C Schatz, Michael R Crusoe, Miguel Roncoroni, Natalie Kucher, Nate Coraor, Nicholas Stoler, Nick Rhodes, Nicola Soranzo, Niko Pinter, Nuwan A Goonasekera, Pablo A Moreno, Pavankumar Videm, Petera Melanie, Pietro Mandreoli, Pratik D Jagtap, Qiang Gu, Ralf J M Weber, Ross Lazarus, Ruben H P Vorderman, Saskia Hiltemann, Sergey Golitsynskiy, Shilpa Garg, Simon A Bray, Simon L Gladman, Simone Leo, Subina P Mehta, Timothy J Griffin, Vahid Jalili, Vandenbrouck Yves, Victor Wen, Vijay K Nagampalli, Wendi A Bacon, Willem De Koning, Wolfgang Maier, and Peter J Briggs. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2022 update. *Nucleic Acids Research*, 50(W1):W345–W351, July 2022.

[37] Maarten Van Steen and Andrew S. Tanenbaum. A brief introduction to distributed systems. *Computing*, 98(10):967–1009, October 2016.

Appendix

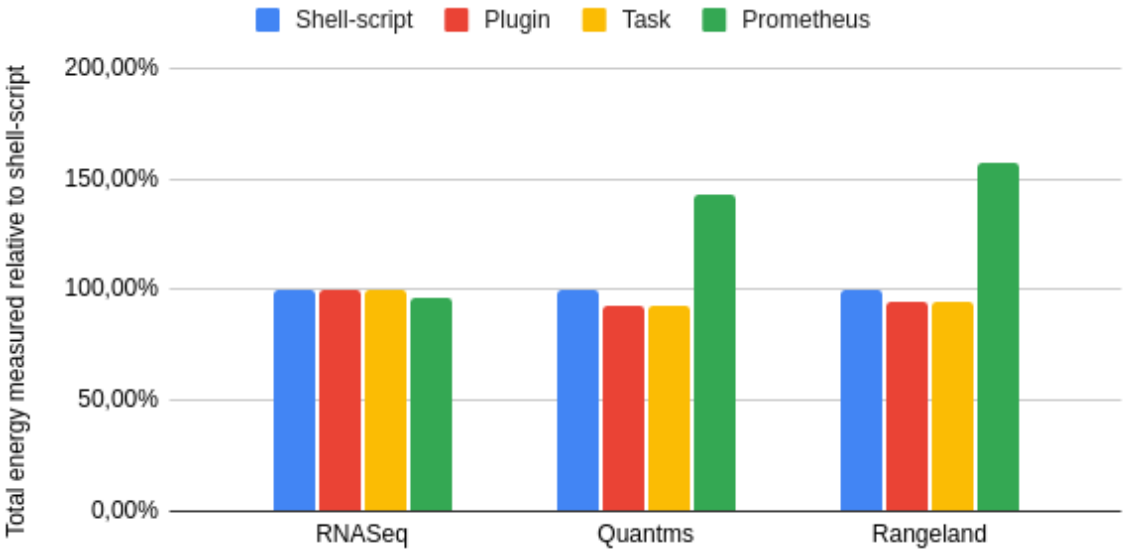

Figure 9: The bar chart shows the differences in the total amount of energy measured using a plugin, an additional workflow task and Prometheus in relation to the amount measured using a shell-script. All values are provided as a percentage of the energy measured with a shell-script, since this method allows starting the measurement before the workflow is initialized. This enables capturing the full energy consumption of the workflow that is measurable with RAPL.

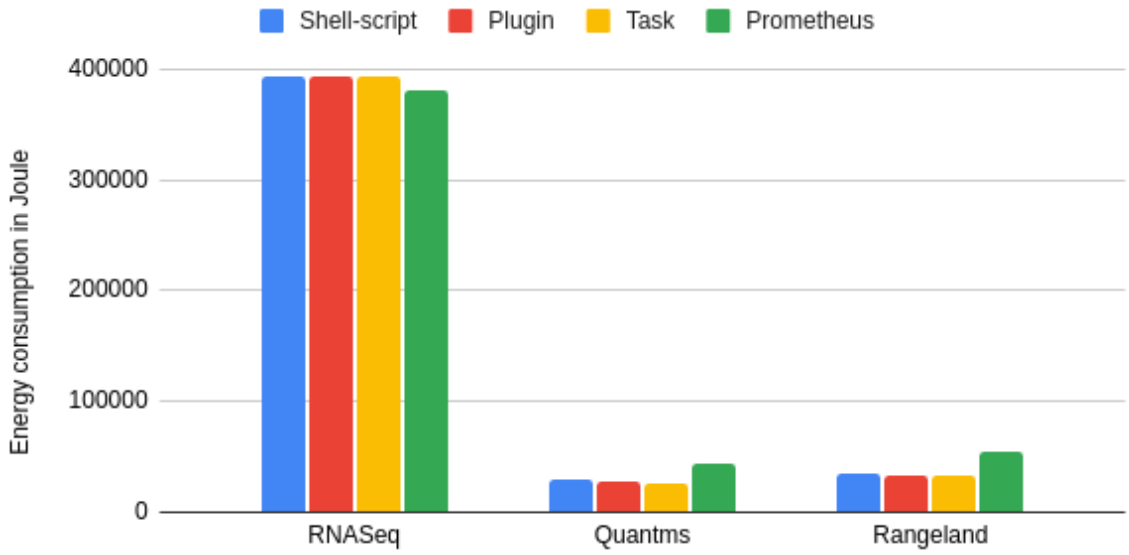

Figure 10: The bar chart shows the total amount of energy measured using the four approaches on a set of three real-world workflows. The numbers presented are averages over five individual runs for each approach on each workflow.