TaskFlow: An Energy- and Makespan-Aware Task Placement Policy for Workflow Scheduling through Delay Management

Laurens Versluis
l.f.d.versluis@vu.nl
Vrije Universiteit Amsterdam
The Netherlands

Alexandru Iosup
a.iosup@vu.nl
Vrije Universiteit Amsterdam
The Netherlands

ABSTRACT

Datacenters need to become more power efficient for political and climate reasons. In this work, we introduce an idea for the community to further explore. We test the idea via an example policy which we call TaskFlow: a makespan conservative, energy-aware task placement policy for workflow scheduling. Using static, rough numbers and simulation, we obtain energy savings between [4.24, 47.00]% and [0.1, 13.6]%, respectively. We also present some pitfalls that should be investigated further, notably starvation of large tasks when using TaskFlow.

KEYWORDS

energy, energy efficiency, workflow, scheduling, cloud, resource, cluster, datacenter, slack, conservative

ACM Reference Format:
Laurens Versluis and Alexandru Iosup. 2022. TaskFlow: An Energy- and Makespan-Aware Task Placement Policy for Workflow Scheduling through Delay Management. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering (ICPE '22 Companion), April 9–13, 2022, Bejing, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3491204.3527466

1 INTRODUCTION

Energy consumption is increasingly the focus of optimization for the cloud sector. As the number of datacenters and our need for computational capacity is rising, our society needs energy-saving innovations in all aspects of cloud operations, from hardware to software. The latter includes scheduling workloads, which is the focus of this work. Current approaches focus on making trade-offs by, e.g., using dynamic voltage and frequency scaling (DVFS) to increase makespan and reduce power consumption [1]. This causes schedulers to pay attention to non-functional requirements (NFRs) such as deadlines and/or budgets, as resources rented trough Infrastructure as a Service (IaaS) providers work on a fine-grained, time and financial, budget. Managing these elements increases scheduling complexity, and hampers adoption in practice due to concerns about, e.g., its effects on fairness [2]. As workloads increasingly consist of workflows [3], the complexity introduced by the inter-dependent tasks (see § 2.1 for a common model) could become unmanageable. Addressing the complex trade-off, in this work we design and evaluate TaskFlow, a relatively low-complexity approach for energy- and makespan-aware workflow scheduling in cloud-like environments.

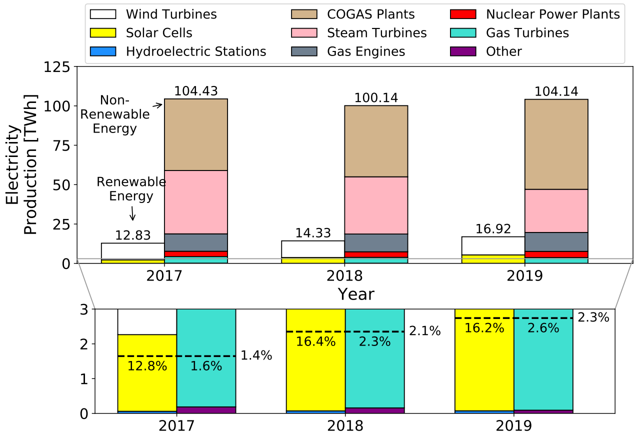

Figure 1: Electricity production in The Netherlands between 2017-2019 grouped by renewable and non-renewable energy sources. The bottom part zooms in and annotates the total energy consumption of all *datacenters* in the country. For both groups, we show the percentage of power being consumed by the datacenters, with the overall consumption percentage to the right. Data from cbs.nl.

Energy consumption has a direct impact on Earth's climate. The ICT sector is already responsible for more than 4% of the global electricity consumption and could increase to an estimated 20.9% in 2030 [4]. Datacenters have an outsized contribution, with an estimated 3-13% of global power consumption by 2030 [5]. With increasing computational needs, e.g., the goal of exascale supercomputers, additional energy savings are imperatively required; it is estimated we need to reach 50 Gigaflops/Watt [6] to keep energy consumption within reasonable limits.

Besides practical limits, there are also political sides to the energy consumption of datacenters. Taking the Netherlands as an example representative of digitalized economies, the public demands efficiency and overall accountability of energy use. Correspondingly, the Central Bureau for Statistics (*Het Centraal Bureau voor de Statistiek*, CBS) collects and publishes regularly information about energy production and consumption. Figure 1 shows an analysis we conduct on such data, collating and depicting multiple datasets published by the CBS regarding the period 2017-2019 (CBS will release soon its final, rather than merely preliminary, data for 2020).

ICPE '22 Companion, April 9–13, 2022, Bejing, China

ACM ISBN 978-1-4503-9159-7/22/04…$15.00
https://doi.org/10.1145/3491204.3527466

The top of the figure depicts the electricity *production* in the Netherlands, by renewability of the source (two stacks, one for renewable, another for non-renewable), and by detailed source (see legend). The bottom of the figure zooms into the first 3% and indicates, through horizontal segments, the fraction of electricity *consumed* by datacenters. We make three main observations. First, *datacenter power consumption shows a rapidly increasing trend.* Already in 2017, datacenters consumed 1.4% of all electricity production. This level of consumption increases yearly by double-digit percentage points, to 2.3% in 2019 (2.7 TWh). The forecast for 2020 was that this percentage would grow to ≈ 3% [7], a prediction matched by preliminary data that indicate growth to 2.8% [8] (3.2 TWh). By 2030, the official body mandated by the Dutch Government (*De Rijksoverheid*) projects further increases to nearly 20% (14 TWh), mainly due to plans for an additional 20-25 datacenters [9–11].

Second, *non-renewable energy production isn't increasing.* The non-renewable energy sources produced roughly 104 TWh in 2017 and 2019, with a small dip in 2018. To avoid requiring additional energy from non-renewable sources to fuel datacenters, one approach is to increase the renewable energy production, which seem to happen at an accelerated pace. The period 2017-2019 shows an increase of 33.6% in renewable energy production. Preliminary data for 2020 indicate further rapid growth of this percentage, to 191.5% compared to 2017 (a 43.3% increase compared to 2019) [12][1].

Third and last, *datacenters consume a sizable chunk of the renewable energy pool*: 12.8% in 2017 and 16.2% in 2019. At the same time, the Dutch government is taking measures to meet the Paris climate agreement, e.g., by enabling more households to become greener. As renewable energy is currently subsidized by the Dutch government, the situation where cloud datacenters consume much of the electricity generated by the new subsidized projects is leading, justly or not, to much public resistance (especially against datacenters built by Amazon, Microsoft, Google, and Facebook) [7, 10, 13, 14].

To match increasing computing demands yet reduce these issues, cloud datacenters need to consume electricity much more efficiently. Various directions to reduce datacenter power consumption are being explored in parallel. These directions include using specialized, power-efficient hardware [15, 16], making use of the already present heterogeneity [1], powering down components [1, 17], improve chip instruction sets [18], etc.

In this work, we focus on energy efficiency achieved by improving the energy-awareness of the *workflow management systems (WMSs)* often found in datacenters. Applications structured as *workflows* are widely adopted in cloud and clusters environments, supporting domains ranging from business critical applications to bioinformatics, and from data analytics to machine learning [3]. Consequently, WMSs are required to manage up to millions of incoming jobs, from up to thousands of users. They operate at a high-level in the datacenter software stack, being able to use the different hardware resources and their software capabilities. WMSs are designed to maximize performance based on one or more metrics. Additionally, Quality of Service (QoS) requirements set by the users, which are typically composed of both NFRs and functional requirements (FRs), form additional constraints that must be met.

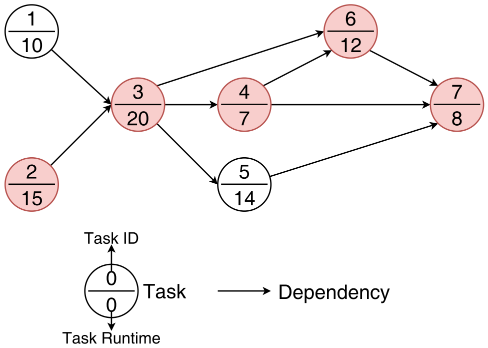

Figure 2: A workflow example. Tasks with darker (red) background form the critical path (see § 2.1).

In this work we present our idea embedded in a task placement policy called TaskFlow. This policy exploits the structure of workflows to reduce the overall power consumption while aiming to avoid affecting the workflow's makespan. Our main objective with this work is to introduce the core ideas and preliminary evidence for the community to explore further. To this end, we make three main contributions:

1. We introduce TaskFlow, which uses a new method to reduce energy consumption for workflow execution (§ 2). TaskFlow takes a conservative approach, leveraging workflow structure to identify *slack*, and in turn using slack to improve power efficiency while avoiding impact on the makespan. This approach limits runtime and memory complexity to $O(|V| + |E|)$, where V is the set of tasks, and E is the set of their inter-dependencies. Thus, this approach works for scheduling both online (at runtime) and offline (before execution).
2. We analyze the effectiveness of using slack to become more power efficient (§ 3). Using realistic traces from three different domains and optimistic execution models, we compute the potential gain of using DVFS and exploiting resource heterogeneity.
3. We analyze the performance of TaskFlow more realistically through extensive simulations (§ 4). Analyzing each workflow on its own through static analysis yields an upper bound. However, in real systems, changing on which machine a task is executed potentially affects other executions. Through simulations, we investigate this cascade.

2 SLACK IN WORKFLOWS

In this section we provide background in workflow scheduling and explain our definition of slack. Next, we discuss how slack can be exploited by introducing several directions.

2.1 Workflows and their Critical Paths

We use the workflow model of Coffman and Graham [19]. In this model, a workflow is composed as a Directed Acyclic Graph (DAG) = (V, E), where V is the set of vertices and E is the set of edges, respectively. Each vertex represents a task and each edge a computation or data constraint. Figure 2 depicts a synthetic, exemplary workflow. A start-node is a task without any incoming edge, which

[1] After an EU agreement, biomass is now labeled as green energy. The numbers in this paragraph are based on this change. The other numbers in this section are based on the old standard.

Table 1: Pros and cons of look ahead for identifying slack.

Pros	Cons
(1) Simple heuristic algorithm. (2) Low runtime complexity. (3) Low space complexity. (4) Easy to parallelize.	(1) Requires knowledge of the structure and arrival times of the direct parents a priori. (2) Does not work for Bags-of-Tasks workflows.

can thus start immediately. An end-node is a task without any outgoing edge, and thus marks the end of a computational chain of tasks. In our example, there are two start-nodes and one end-node.

The *critical path* defines the path(s) from any start node to any end node that forms the largest sum of runtimes (including wait times such as data (estimated) transfer times etc.). In our example workflow the vertices marked red denote the critical path. Assuming enough resources and no abnormalities, the critical path dictates the total runtime of the workflow, 62 time units in our example.

2.2 Parallelism and Slack in Workflows

In workflow execution, it is common that multiple (chains of) tasks can be executed in parallel. In prior work we demonstrated, through extensive characterization of real-world execution traces, that most workloads exhibit parallelism [3]. The idea is to exploit the difference in task runtimes. For simplicity, we assume data transfers times are included in the task runtimes. In our example workflow, there are two parallel chains at the start. Both have length 1 (a single task) and end at Task 3. Task 1 has a runtime of 10, whereas the runtime of Task 2 is 15; we can delay Task 1 by up to 5 seconds before becoming the critical path of the workflows. We refer to this room for delay as *slack*.

The idea we leverage in this work is to use slack to reduce power consumption by running tasks that can be delayed on slower, but more power efficient hardware, or slow down the runtime by, e.g., using DVFS techniques. As we only use slack, the tasks on the critical path are not slowed down, which causes the critical path to remain the same. Thus, the *workflow makespan*, which is the time elapsed since the start of the workflow until the completion of its last remaining task, remains unchanged—a common QoS-goal.

2.3 Identifying Slack

To identify slack in a workflow, we use in this work a look-ahead approach that has been used in prior work [20, 21]. The core operation is to compute the earliest possible start time of each task based on the runtimes of all its parents, recursively. For each task, we then obtain the available slack by computing the difference between the earliest finish time of this task and the earliest start time of all its children. We outline the computation of the start times in Algorithm 1 and the assigning of slack in Algorithm 2. Topological sorting of a graph has a runtime of $O(|V| + |E|)$ which was proven by Kahn [22]. The computational runtime of the topological sorting can be lowered through the use of parallel and distributed computing, which would help in a distributed computing environment [23]. As we loop over the topological order, the assignment of slack also has a runtime of $O(|V| + |E|)$, a potential improvement over the runtime of $O(|V|^2)$ proposed by Li et al. [20]. As the workflow is kept in memory and only a mapping of slack per task is required to be maintained, the space complexity for both algorithms is $O(|V| + |E|)$.

Algorithm 1: Minimal task start times computation. Runtime: $O(|V| + |E|)$ Space: $O(|V| + |E|)$

```
Input: A workflow $w$
Result: A map with the lowest possible start time of each
        task in $w$
$\gamma \leftarrow$ map of task arrival times;
$\zeta \leftarrow$ map of task runtimes;
foreach wave $\omega$ in topological sorting of $w$ do
    foreach task $t$ in $\omega$ do
        $ft \leftarrow \gamma[t] + \zeta[t]$;
        foreach child $c$ of $t$ do
            if $ft > \gamma[c]$ then
                $\gamma[c] \leftarrow ft$;
            end
        end
    end
end
return $\gamma$;
```

Algorithm 2: Assigning slack after topological sorting. Runtime: $O(|V| + |E|)$ Space: $O(|V| + |E|)$

```
Input: Map of minimal start times $\gamma$ from Algorithm 1 and
       workflow $w$
Result: Mapping of slack per workflow
initialize slack as empty map;
foreach task $t$ in $w$ do
    if $t$ has no children then
        $slack[t] = 0$;
    else
        $c_{min} = min\{\gamma[c]$ for each child $c$ of $t\}$;
        $slack[t] = c_{min} - \gamma[t]+$ runtime of $t$;
    end
end
return slack;
```

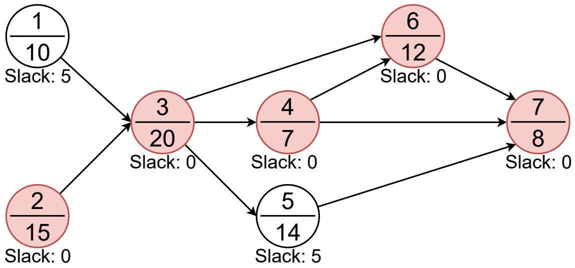

Figure 3: The example workflow from Figure 2, with "Slack" annotations for the amount of slack identified per task using the look-ahead method (see § 2.3).

Table 2: Task slack in milliseconds per percentile per domain.

	1^{st}	25^{th}	50^{th}	mean	75^{th}	99^{th}
Engineering	0	0	23,113	92,903	76,621	1,140,122
Industrial	0	9,000	31,000	136,728	101,000	1,987,000
Scientific	0	0	80	169,695	1,952	3,200,659

To illustrate the outcome of our look ahead algorithm, we apply it visually to our example workflow where we assume all tasks arrive at time 0, see Figure 3. As explained before, the two chains at the start both have task 3 as a child. Hence, the earliest start time of Task 3 is at time 15. As Task 1 has a runtime of 10, it gets a slack assigned of 5: its parents minimal start time is 15 due to Task 2, and its own start time plus runtime equals 10, a difference of 5. Using similar reasoning, we obtain a slack of 5 for Task 5.

Next, we investigate the presence of slack in real-world IT operations. We apply for this purpose the look-ahead algorithm to the traces hosted by the Workflow Trace Archive (WTA) [3]. The WTA is a high-quality, curated, community archive of real-world traces of workloads of workflows. From the Alibaba trace provided by the WTA, we use the first 100,000 workflows (31.6 million tasks) sorted by their IDs as the entire dataset is too large to analyze without large computing resources.

Table 2 summarizes the results, tabulating by application domain, using WTA-provided labels. We observe that *workflows across all domains contain slack*. Engineering workloads offer the lowest slack on average and at the tail-levels. Although scientific workflows have the highest slack on average, they have the lowest median value: 50% of all tasks having a slack of 80 ms or lower, indicating scientific workloads have the heaviest tail and the fewest possibilities to reduce power consumption using our conservative approach. Industrial tasks have the highest median slack and the second-highest $99th$-percentile value. We will investigate next to what extent we can use slack with different distribution to improve energy consumption.

3 USING SLACK TO REDUCE ENERGY CONSUMPTION

In this section, we consider two methods that can use slack identified by TaskFlow to reduce energy consumption:

(1) Slow down tasks using dynamic voltage and frequency scaling (DVFS).
(2) Schedule tasks on potentially slower, yet more power efficient hardware.

We investigate the potential gains these approaches offer through static analysis of the tasks and their slack. In this analysis, we assume plenty of resources are available and that all support DVFS. It is unlikely that this assumption capture reality well, as this analysis assumes that delaying a task has no impact on any other task. Nonetheless, by using this assumption we can provide an optimistic bound of the effects of using slack to reduce energy-consumption. This type of analysis aligns with that of, among others, [1, 21, 24, 25], which all assume resources are infinite or that additional resources can always be obtained from (external) cloud environments.

Table 3: Delay and energy savings of four DVFS settings. Data from [26].

ID	1	2	3	4
Delay	0%	22.86%	53.44%	147.28%
Energy Savings	0%	8.6%	12.60%	12.40%

Table 4: Average energy reduction per domain using DVFS.

Domain	Engineering	Industrial	Scientific
Energy reduction	7.62%	11.35%	4.24%
Overall average		11.31%	

3.1 Using DVFS

Using DVFS, the voltage and/or frequency of a machine is lowered to reduce its power consumption at the cost of processing speed. While the returns are diminishing with modern hardware [27] and results vary per workload, there are still scenarios where power savings are possible. Tasks that have slack available can thus potentially be delayed to save some energy.

Few articles present numbers in terms of delay and energy saved. We couldn't find any article that offers both their code, setup, and datasets used in experimentation. To this end, we use the numbers reported by Dhiman et al. [26]. The authors use single-threaded and multi-threaded workloads using four DVFS settings. We use (and assume they hold) their multi-threaded results ([26], table 3) and average the numbers as our workloads are likely a mix of multiple applications, see Table 3. From this table, we observe that the delay increases substantially versus energy saved. Moreover, we note that the last settings does not yield any benefit on average compared to the third option.

Using the numbers in Table 3, we investigate how much energy can be saved, see Table 4. We select the highest delay factor from Table 3 that is less than or equal to $\frac{t_r+t_s}{t_r}$ for each task where t_r is the runtime and t_s is the slack. We omit tasks that have a runtime of 0 as they could theoretically be delayed indefinitely. From this table we observe that across all domains, we can achieve a 11.31% energy reduction by trading off slack using DVFS. This number is skewed in favor of industrial tasks due to the large number of these tasks in our analysis. The industrial domain shows the most potential gain on average at 11.35%. This matches our prior findings where industrial tasks have the most slack on average and by median. The engineering domain has the second highest gain on average at 7.62%, which also corresponds to our earlier findings. Scientific tasks show the least average potential gain at 4.24%.

3.2 Using Heterogeneity

Next, we look at exploiting heterogeneity of hardware. Using slack, we might be able to run tasks on less powerful yet more power efficient nodes when available. Some nodes might even offer this themselves by using, e.g., the big.Little configuration [28, 29] Similar to DVFS, we trade computational speed for power, yet do not alter configurations of hardware. We do assume here that all tasks fully utilize the required cores, and that energy consumption is proportional to the number of cores used and the machine's Thermal Design Power (TDP). This assumption does not hold in general as

Table 5: Power consumption statistics for some of the latest CPU models.

Model	Base Clock [GHz]	Logical Cores	TDP [W]	W/GHz/ core
AMD Ryzen Threadripper 3990X	2.9	128	280	0,75
AMD Ryzen 9 5900X	3.7	24	105	1,18
Intel i9-10900K	3.7	20	125	1,69
AMD Ryzen 7 5800X	3.8	16	105	1,73
Intel i7-10700K	3.8	16	125	2,06
AMD Ryzen 5 5600X	3.7	12	65	1,46
Intel i5-10600K	4.1	12	95	1,93

Table 6: Average energy reduction per domain using heterogeneity.

Domain	Engineering	Industrial	Scientific
Energy reduction	28.31%	41.61%	16.68%
Overall average		41.47%	

the available I/O, memory, and storage impact task performance. However, if power normalization [15] becomes a reality these assumptions would hold.

Table 5 shows six recent CPU models. In this table we list the base clock speed, number of cores, and TDP reported by the respective manufacturers. We assume TDP is the upper limit as these values are not exceeded in modern HPC clusters [30]. Assuming power normalization, we compute the Watt per GHz per core. From this table, we observe clear differences when comparing power per GHz per core, with the lowest being the Threadripper 3990X at 0,75 and the highest the i7-10700K at 2,06, an increase of 2,75×.

We take for each distinct clock speed the best performing model based on the Watt usage per GHz per core. We assume all task runtimes are based on the highest base clock and scale linearly in clock speed. Additionally, we assume large tasks can be split across multiple machines (of the same type). For each task, we select the machine with the lowest W/GHz/core ratio such that $\frac{t_r+t_s}{t_r} \leq \frac{bc_{max}}{bc_{current}}$ where bc_{max} is the highest base clock speed of any machine where the task can run on and $bc_{current}$ is the base clock speed of the machine being checked. The results are in Table 6.

From this table, we observe significantly higher gains than using DVFS with an overall average reduction of 41.47%. This makes sense as the delay observed in base clock is at most 1.41× with energy gains of up to 2.75×. Again, the industrial domains contains the most slack and hence benefits the most through heterogeneity, on average a reduction of 41.61%. For the scientific and engineering domains, this reduction is 28.31% and 16.68%, respectively. While optimistic and an upper bound, the results underlines the power efficiency of different systems. The results align with the findings of similar studies that present even higher gains by using non-conservative approaches, e.g., [1, 25].

3.3 Using heterogeneity and DVFS

Finally, we investigate if combining these two techniques can lead to a higher efficiency, theoretically. We apply the machine selection approach of Section 3.2 and attempt to use DVFS to further consume any leftover slack. The results are in Table 7. From this table, we observe DVFS indeed can consume some leftover slack, rising the overall average energy reduction from 41.47% to 46.85%. We observe additional reductions of 3,80%, 5,39%, and 1,77% for the engineering, industrial, and scientific domains, respectively. While the gains are diminishing, it does show that these approaches, on paper, can be combined to increase efficiency.

Table 7: Average energy reduction per domain using both heterogeneity and DVFS.

Domain	Engineering	Industrial	Scientific
Energy reduction	32.11%	47.00%	18.45%
Overall average		46.85%	

4 SIMULATION EXPERIMENTS

In this section we perform several experiments through simulation. We assumed prior that identified slack can be trade-off without consequence and that machines of each model are available without limit. In real systems this is rarely the case. Resources are finite and it might be that machines of a certain type are completely booked, forcing us to make a less desirable choice.

Increasing the runtime of a task has consequences on the runtime (and slack!) of tasks waiting in the queue. While these cascading effects are not taken into account in our static analysis, we do potentially run into these effects during simulation. Via simulations we can investigate this effect and check to what extent TaskFlow is indeed makespan conservative. We opt for simulation to replay workloads that would otherwise take significant resources and time to execute in emulation or in a real-world setting.

4.1 Experimental Setup

Next, we describe our experimental setup per component.

Simulator. For our simulations, we use a discrete event simulator that we used in our prior work [3]. We expand the simulator by adding a baseline policy, TaskFlow, and required code for energy analysis. The simulator uses a backfilling approach where it attempts to put smaller tasks on available slots if the task at the head of the queue is too large. Important to note is that due to the complexities mentioned earlier, TaskFlow makes a best effort approach for scheduling tasks on resources[2]. If there are insufficient resources to run a task before its parents' start times, but there are enough (slower) resources available to schedule this task right now, TaskFlow will use those resources. Determining if sufficient resources become available in the near future so that the task does not delay its parents will significantly increase the complexity of the policy, which would decrease its adoptability.

Workloads. We use all traces from the WTA [3] that feature DAG-based workflows. Some of these traces features large bursts of task arrivals at the start, causing tasks to queue, reducing their slack. Additionally, the resource requirement vary significantly between these traces, up to 1,000 resources per task. Workflows and their tasks arriving at identical times are ordered on their respective identifiers to make our experiments reproducible when running our policies.

[2] We use the generic term resources as the original tasks required different resources, e.g., threads, CPU (cores), VMs, etc.

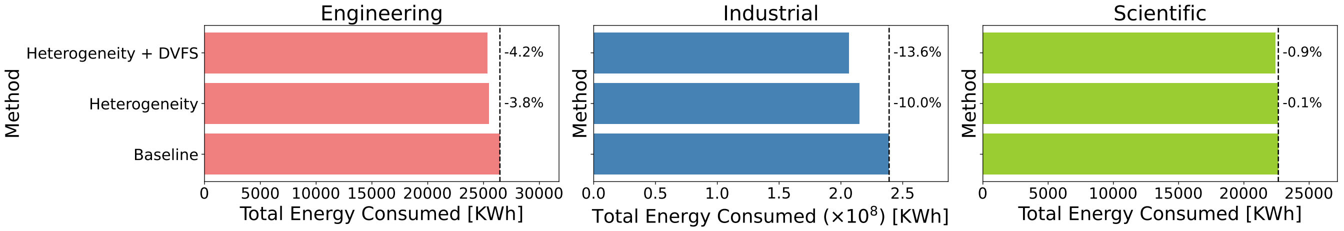

Figure 4: Total *energy consumption* per domain using the baseline and TaskFlow exploiting heterogeneity with and without DVFS enabled.

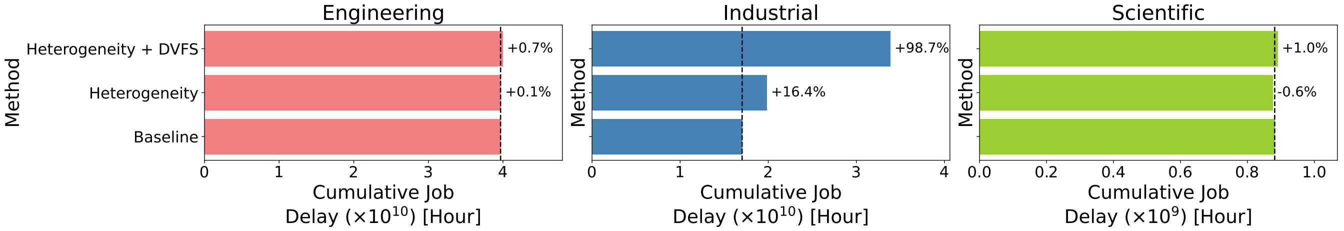

Figure 5: The *cumulative workflow delay* per domain for the baseline and TaskFlow exploiting heterogeneity with and without DVFS enabled.

Table 8: Workflow delay in minutes per percentile for the Alibaba workload using TaskFlow with DVFS enabled.

	1%	25%	50%	75%	90%	99%	100%
Delay	0.00	0.00	0.00	3.11	28.13	168.02	560.27

The computing environment. We use two different machine models, the fastest (by base clock) and most power efficient from Table 5. To challenge our policy, for each workload, a static environment is provisioned based on a 40% utilization rate. This utilization rate matches hyper-scale cloud computing environments [31]. To determine the utilization rate, for each workload, we compute the average resource seconds required by dividing the total resource seconds in the workload by its length.The number of machines per model is set to serve half of the workload with a minimum of 1 for each to enforce heterogeneity.

The baseline. To measure the impact of TaskFlow, we use a greedy policy as baseline. It focuses on throughput by scheduling each task on the fastest machine available.

Assumptions. We assume that data transfer times are included in the task runtimes and that tasks scale with the base clock speed of machines. Tasks are not rescheduled once placed. Next, we assume that machines are energy-proportional and that we can apply DVFS instantly to specific cores, matching [21, 25]. We only track energy consumed by executing tasks, i.e., idle time is not included. Tasks with a runtime of 0 are considered having a runtime of 1.

4.2 Experimental Results

To measure the performance of TaskFlow, we compare the total energy consumption and cumulative job delay per domain with that of the baseline policy. We run TaskFlow with and without DVFS enabled to observe the additional impact DVFS has after exploiting the available heterogeneity.

Figure 4 shows the total energy consumption per domain for the three settings, while Figure 5 shows the amount of time that all workflows in each workload were delayed in respect to their critical paths. From these figures we observe that TaskFlow has the lowest impact on Scientific workloads, reducing energy consumption by 0.1% solely using heterogeneity and by 0.9% through the addition of DVFS. The change in workflow delay is also minimal: between [-0.6, 1]%. Engineering workflows have a larger power reduction, between [3.8, 4.2]% at the cost of 0.1% and 0.7% additional delay, respectively. The industrial workloads show significantly different behavior. Their power reductions are significantly higher, with 10.0% solely using heterogeneity and 13.6% using heterogeneity and DVFS. However, the cumulative delay increases substantially, with 16.4% and 98.7%, respectively. We observe tasks in the Alibaba workload are starved, mainly due our backfilling approach. The additional delay introduced by DVFS could cause more jobs to enter the system in the meantime that also get backfilled, additionally starving the large tasks, increasing the delay to 98.7%. Table 8 shows the delay per workflow for the Alibaba trace. We observe from this table that workflow delays show long tail behavior, pointing to a few workflows causing the majority of the delay. This finding was surprising to us and underlines the importance of investigating all aspects of the scheduler, as we discuss in our prior work [32]. Even though our policy focuses on being conservative, cascading effects can lead to significant impacts. We note that other policies using similar performance-energy trade-off techniques likely face similar issues when starvation occurs, and that techniques such as fair scheduling could solve this particular problem.

As the total sum does not sketch the full picture, we present in Figures 6 and 7 boxplots of the workflow delay per workflow, grouped by domain and method. From these figures we observe barely any difference in workflow delay for the scientific and engineering domain, as expected. The industrial domain shows for solely using heterogeneity a very similar pattern to the baseline, only showing an increase in delay at the long tail, as observed earlier. The combination of heterogeneity and DVFS does significantly increase the delay and the number of workflows delaying. There thus seems to be a point where delaying tasks start to increase the queue, delaying workflows that shouldn't be delayed.

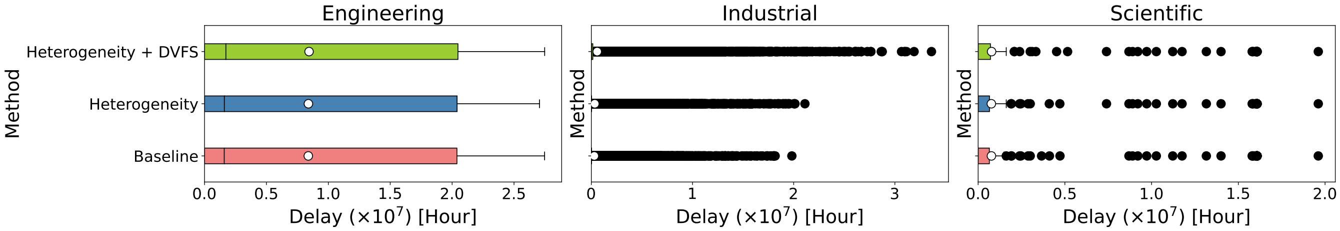

Figure 6: *Workflow delay* per domain for the baseline and TaskFlow exploiting heterogeneity with and without DVFS enabled. The white dot depicts the average.

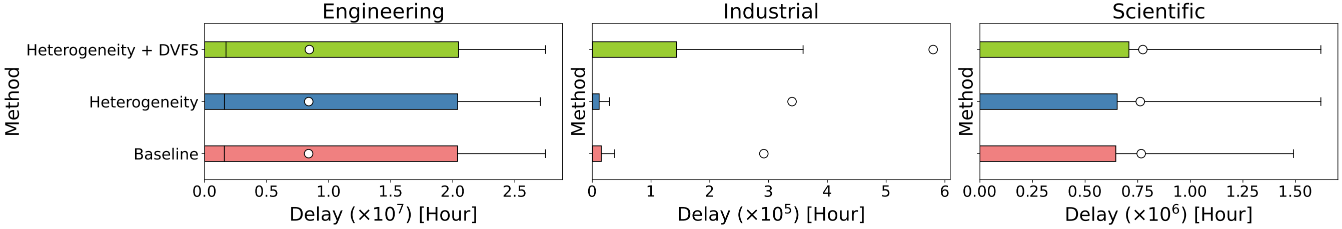

Figure 7: The results values presented in Figure 6 with outliers removed. The original averages are retained.

5 RELATED WORK

In this section, we cover related work on energy-aware workflow scheduling. We focus on articles that use conservative approaches and slack. We discuss our contributions with respect to each one. Overall, *none* of the related work using simulation use the extensive suite of traces that we do.

Conservative approaches: Few articles use a conservative approach. Xu et al. [33] focus on pipeline workflows (sequential workflows), which this work extends to the general notion of DAG-based workflows. Second, they consider solely machine placement whereas we also investigate the effect of DVFS.

Trading slack: Closest to our work is Xie et al. [17]. Their approach focuses on turning of as much hardware as possible and then check if delaying concurrent running tasks is possible. Their approach is non-conservative as the critical path of a workflow is extended (see, e.g., Figure 6 in their work). Second, their method of computing slack differs. By looking at concurrent running tasks, its possible slack is missed that would otherwise be found by TaskFlow. Li et al. [20] propose another method, where their slack computation is identical. Their algorithm for slack detection requires $O(|V|^2)$, whereas ours requires $O(|V| + |E|)$. Our work additionally complements theirs by considering the heterogeneity of the physical hardware, where they consider VMs. Piteri et al. [34] introduce an offline scheduling approach. They first use HEFT to create a schedule plan and then check which machines can be delayed through DVFS without exceeding a deadline to save energy. Tang et al. [25] extend the work of [34]. Different from our work, their work employs an offline approach. First, a plan is created using HEFT and the workflow makespans are based on that. Next, workflows are delayed to stay below a defined deadline. Medara et al. [21] use a similar approach to [25], yet take into consideration network energy costs next to improving the reliability of the system by taking into account the possible errors caused by DVFS. Wiesner et al. introduce a method to use different regions where it’s likely to reduce emissions, at the expense of delaying workloads [35]. In their model they take power grid fluctuations and green energy into consideration.

6 THREATS TO VALIDITY

In this section we discuss the main threats we perceive to the validity of this work.

The first threat is that we assume knowledge of task runtimes. Prior work has already demonstrated that even with very simple heuristics, one get reasonable task runtime estimates [36]. More complex predictors yield even better results. Moreover, this work fits very well with WMSs such as Airflow. Workflows in these environments can be executed periodically, akin to cronjobs. Parsing, e.g., daily logs or performing routine checks can be predicted very well. One of the biggest threats to this model is performance variability, where identical tasks using identical input on identical hardware vary in runtime due to multi-tenancy or other factors [37]. Such variability could be mitigated by lowering the found slack by a certain factor, to allow for additional uncertainty. However, overall, we believe using estimated runtimes is feasible.

Another threat is requiring the structures of the workflows beforehand. It’s possible that structures are unknown for streaming or dynamic workflows. We acknowledge that this poses limitations, however, our method could be used on groups of tasks that only share common parents. Tasks with a lower runtime than the maximum in such a group can never be the critical path, even if this parent is part of the critical path. Hence, the entire structure of a workflow need not to be known then. Yet, this might reduce the amount of slack identified. Second, there are frameworks and products that explicitly operate on entire DAGs. Spark and Apache Airflow are two examples of applications where the entire DAG is known beforehand. Finally, one could make use of predictions to guess the structure of the workflow, based on past executions of streaming or dynamic workflows.

A third threat is CPU usage and energy proportionality. In this work, we assume that tasks fully utilize their assigned resources and that energy consumption is proportional to the number of cores used. Currently, most machines have their top energy efficiency at 100% CPU utilization [38], with 70% to 100% being the best range for almost all systems. However, this landscape is changing with providers such as Google pushing for energy proportionality to become a primary design goal across the entire spectrum, including network [16] and CPU power consumption [15]. TaskFlow is trivially adaptable to the current energy efficiency profiles, yet would require information on the (predicted) resource utilization.

A fourth threat is that for the industrial workloads only the Alibaba and Shell traces qualify. The other industrial traces in the WTA do not express task dependencies. We cannot overcome this limitation unfortunately, as the disclosure of realistic industrial traces is at the discretion of industrial companies. Already, in our prior work [3] we advocate for companies to release more traces, as approaches such as this work can then be validated on their workloads, which is also to their benefit.

A fifth threat is that we do not consider making trade-offs when putting machines into idle or sleep mode. In this work, our main goal is to posit this technique and demonstrate its performance. Le Sueur and Heiser show that in certain cases introducing idle modes improves energy savings when using DVFS [27]. Thus, our method should be even more efficient in such scenarios. We therefore do not perceive this as a threat to the validity of the work, yet as future work to explore the full potential of this technique.

7 CONCLUSION AND ONGOING WORK

Addressing the increasingly pressing need to make cloud operations more energy-efficient, we focus in this work on the workflow scheduling component. We leverage the idea of using the workflow structure to identify tasks that can be delayed without impacting the critical path. We then investigate two techniques that can use this form of slack to run the workflow more energy-efficiently. Using first optimistic analysis and then detailed simulation, we provide support for further investigating this idea and identify parts that require immediate attention. We also signal that the problem is not trivial—if tasks are starved, such an approach can actually make the situation worse.

We hope that the community finds additional ways to exploit slack. In the future, together with the community, we aim to investigate how well TaskFlow performs with an anti-starvation policy or backfilling disabled, and, further, how this idea affects various NFRs. Costs is a popular NFR; slack could influence cost-based decisions as different resources can have different costs depending on the cost model. Moreover, providers violating Service Level Objectives (SLOs) often pay a penalty to the affected client, it could be studied how often a policy like TaskFlow would violate SLOs. Other NFRs such as fault-tolerance through preemptively running speculative copies in identified slack-gaps might be worthwhile to investigate.

REPRODUCIBILITY AND OPEN-ACCESS DATA

To support reproducible science, we offer the software used to analyze as open-source data at https://github.com/atlarge-research/TaskFlow-software. The simulator and traces are available at https://github.com/atlarge-research/wta-sim/tree/tasks-across-machines and https://wta.atlarge-research.com, respectively.

REFERENCES

[1] Cao *et al.*, "Energy-efficient resource management for scientific workflows in clouds," in *SERVICES*, 2014.

[2] Klusácek and Tóth, "On interactions among scheduling policies: Finding efficient queue setup using high-resolution simulations," in *Euro-Par*, vol. 8632, 2014.

[3] Versluis *et al.*, "The workflow trace archive: Open-access data from public and private computing infrastructures," *TPDS*, vol. 31, 2020.

[4] Nature, "How to stop data centres from gobbling up the world's electricity," https://www.nature.com/articles/d41586-018-06610-y, 2018, accessed: 2021-03-25.

[5] Andrae and Edler, "On global electricity usage of communication technology: trends to 2030," *Challenges*, vol. 6, 2015.

[6] Villa *et al.*, "Scaling the power wall: A path to exascale," in *SC*, 2014.

[7] NRC, "Gebroken beloftes: hoe de wieringermeerpolder dichtslibde met windturbines en datacentra," https://www.nrc.nl/nieuws/2020/06/05/gebroken-beloftes-hoe-de-wieringermeerpolder-dichtslibde-met-windturbines-en-datacentra-a4001882, 2020, accessed: 2021-03-13.

[8] CBS, "Elektriciteit geleverd aan datacenters, 2017-2020," https://www.cbs.nl/nl-nl/maatwerk/2021/50/elektriciteit-geleverd-aan-datacenters-2017-2020, 2021, accessed: 2022-01-22.

[9] Rijksoverheid, "Ruimtelijke strategie datacenters," https://www.rijksoverheid.nl/documenten/rapporten/2019/03/15/ruimtelijke-strategie-datacenters, 2020, accessed: 2021-03-25.

[10] NRC, "Datacentra zeewolde vragen twee keer zoveel stroom amsterdam," https://www.nrc.nl/nieuws/2020/06/21/zeewolde-speelt-straks-champions-league-met-stroomverbruik-a4003548, 2020, accessed: 2021-03-25.

[11] NOS, "Kabinet: nog plannen voor 20 tot 25 datacenters," https://nos.nl/artikel/2409866-kabinet-nog-plannen-voor-20-tot-25-datacenters, 2021, accessed: 2022-01-22.

[12] CBS, "Hernieuwbare elektriciteit; productie en vermogen," https://opendata.cbs.nl/#/CBS/nl/dataset/82610NED/table, 2021, accessed: 2022-01-22.

[13] NRC, "Nieuwe datacenters? het papierwerk komt later wel," https://www.nrc.nl/nieuws/2020/11/24/nieuwe-datacenters-het-papierwerk-komt-later-wel-a4021302, 2020, accessed: 2021-03-25.

[14] ——, "Datacenters zijn in nederland 66 procent meer stroom gaan gebruiken," https://www.nrc.nl/nieuws/2020/12/14/datacenters-zijn-66-procent-meer-stroom-gaan-gebruiken-a4023781, 2020, accessed: 2021-03-25.

[15] Barroso and Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, 2007.

[16] Abts *et al.*, "Energy proportional datacenter networks," in *ISCA*, 2010.

[17] Xie *et al.*, "Energy management for multiple real-time workflows on cyber–physical cloud systems," *FGCS*, vol. 105, 2020.

[18] Tillich and Großschädl, "Instruction set extensions for efficient AES implementation on 32-bit processors," in *CHES*, 2006.

[19] Jr. and Graham, "Optimal scheduling for two-processor systems," *Acta Inf.*, vol. 1, 1972.

[20] Li *et al.*, "Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds," *IEEE TSC*, vol. 11, 2015.

[21] Medara and Singh, "Energy efficient and reliability aware workflow task scheduling in cloud environment," *Wireless Personal Communications*, 2021.

[22] Kahn, "Topological sorting of large networks," *CACM*, vol. 5, 1962.

[23] Sanders *et al.*, *Sequential and Parallel Algorithms and Data Structures - The Basic Toolbox*. Springer, 2019.

[24] Chen *et al.*, "EONS: minimizing energy consumption for executing real-time workflows in virtualized cloud data centers," in *ICPP*, 2016.

[25] Tang *et al.*, "An energy-efficient task scheduling algorithm in dvfs-enabled cloud environment," *Journal of Grid Computing*, vol. 14, 2016.

[26] Dhiman *et al.*, "Analysis of dynamic voltage scaling for system level energy management," *USENIX HotPower*, vol. 8, 2008.

[27] Le and Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems*, 2010.

[28] Padoin *et al.*, "Performance/energy trade-off in scientific computing: the case of ARM big.little and intel sandy bridge," *IET Comput. Digit. Tech.*, vol. 9, 2015.

[29] Yoo *et al.*, "A case for bad big.little switching: how to scale power-performance in SI-HMP," in *HotPower*, 2015.

[30] Patel *et al.*, "What does power consumption behavior of HPC jobs reveal? : Demystifying, quantifying, and predicting power consumption characteristics," in *IPDPS*, 2020.

[31] Whitney and Delforge, "Data center efficiency assessment," *Issue paper on NRDC (The Natural Resource Defense Council)*, 2014.

[32] Andreadis *et al.*, "A reference architecture for datacenter scheduling: Extended technical report," in *SC*, 2018.

[33] Xu *et al.*, "Enreal: An energy-aware resource allocation method for scientific workflow executions in cloud environment," *IEEE TCC*, vol. 4, 2016.

[34] Pietri and Sakellariou, "Energy-aware workflow scheduling using frequency scaling," in *2014 43rd International Conference on Parallel Processing Workshops*, 2014.

[35] P. Wiesner *et al.*, "Let's wait awhile," in *Middleware*. ACM, 2021.

[36] Chirkin *et al.*, "Execution time estimation for workflow scheduling," *FGCS*, 2017.

[37] Uta *et al.*, "Is big data performance reproducible in modern cloud networks?" in *NSDI*, 2020.

[38] Jiang *et al.*, "Energy proportional servers: Where are we in 2016?" in *ICDCS*, 2017.