An Energy-aware Analysis of Provisioning Policies

Herman Kelder
2752644
VU Amsterdam
h.g.kelder@student.vu.nl

Bryan Westerveld
2752643
VU Amsterdam
b.westerveld@student.vu.nl

Skip Thijssen
2792737
VU Amsterdam
s.thijssen@student.vu.nl

Sander Misdorp
2751209
VU Amsterdam
s.j.misdorp@student.vu.nl

Niels Keinke
2649910
VU Amsterdam
n.b.keinke2@student.vu.nl

Peter-Jan Gootzen
2703924
VU Amsterdam
p.j.m.gootzen@vu.nl

ABSTRACT

The increasing complexity and size of computational problems have led to larger and more energy-intensive datacenters. As a result, energy efficiency is a key area of research, particularly in large-scale systems where even small gains can result in significant reductions in energy expenditures. In this report, we implement the TaskFlow energy-aware scheduling algorithm in the OpenDC datacenter simulator to investigate the effect of different provisioning policies on power consumption and resource utilization. Our results indicate that the selection of a provisioning policy significantly impacts the power consumption of datacenter systems, with TaskFlow using less energy per scheduling cycle but taking longer to complete workflows compared to a naive and random scheduler. Overall, the findings demonstrate the importance of considering energy efficiency in scheduling decisions and the usefulness of simulation tools like OpenDC for evaluating the impact of such algorithms.

1 INTRODUCTION

The computational problems that our society needs to solve are ever-increasing in size and complexity, leading to more and bigger datacenters that consume a growing chunk of our society's energy supply [1, 2]. In recent times energy usage has been put forward as one of the key issues of our time because of the current political and environmental issues. In our everyday lives, we see the prices of gas, petrol and electricity rise drastically [3], and these price increases pose significant harm to companies as well [4].

The research community has done much work concerning energy efficiency in many areas, with the focus mainly on the outliers of the spectrum between low-energy availability (e.g. mobile computing) and high-energy usage (e.g. datacenters and scientific computing). The large-scale systems are especially of interest because massive amounts of power can be saved even with marginal efficiency gains. The community has done extensive research on energy optimizations in (among others) networking [5, 6], accelerators [7–9] and cooling [10, 11].

One system design choice which runs throughout the complete spectrum of computing and that plays a vital role in the datacenter is the challenge of scheduling [12–15]. A computational system tries to schedule its work to satisfy its tenants (whether contractually or for the sake of user experience) who can have differing requirements. The effects of a scheduling decision can often not fully be known until the decision has been executed, coupled with the fact that in a high-velocity environment, a slow scheduling decision (regardless of the decision's quality) can negatively impact the overall system performance leading to scheduling is a very complex problem.

There a many approaches to evaluating a scheduling algorithm, the most representative of which is to use the scheduler in a real "production" environment (*in vivo*). This is, however, often not feasible for a datacenter-scale scheduling algorithm as not all researchers have access to such a system, and operators are generally too risk-averse to allow such an experiment. One of the possible experimentation approaches to combat this is to do simulations of real systems and real workloads with said algorithm. The OpenDC simulator project of the AtLarge research group tries to do precisely this [16, 17]. By providing a holistic framework for simulating a real-world datacenter and its workloads, researchers can do large-scale datacenter experimentation with limited resources [18, 19].

In this report, we will further expand on the work of Versluis and Iosup on the TaskFlow energy-aware scheduling algorithm [20]. By implementing the TaskFlow algorithm into the OpenDC datacenter simulator and investigating the effect of changing the provisioning scheduler on the resource utilization and power usage of a simulated datacenter, we hope to extend the capabilities of OpenDC further and attempt to reproduce the results of the TaskFlow paper.

2 BACKGROUND

This section will cover the high-level workings of the simulator that we use called OpenDC, and the workings of the TaskFlow energy-aware scheduling algorithm that we implemented in OpenDC.

2.1 OpenDC

OpenDC is an open-source datacenter simulator framework designed by the AtLarge research group. OpenDC aims to provide the research community with a common base for researching datacenter-like systems and provide the teaching community with an instrument with which graduate students can learn engineering and scientific practices [16]. The simulator models can model datacenters on the level of physical machines, virtual machines and containers, supporting a wide array of workload types, including state-of-the-art workloads such as Machine Learning and serverless [17].

For this report, the workflow trace simulation stands central. A *workflow* is an abstract representation of a single workload that runs inside of datacenters and is defined as a DAG (directed acyclic graph) containing *tasks* (tasks are small pieces of execution with

certain characteristics). A set of workflows is called a *trace* and represents the complete set of workloads that happen in a datacenter over a given period of time. In Figure 1, a single example workflow contains seven tasks with a runtime and possible dependencies that only allow them to run when certain other tasks have finished running. The runtime is based on a particular machine's performance, allowing the runtime to be translated into FLOPs which can be (roughly) scaled to machines of differing performance.

To support open and reproducible research, OpenDC can run workflows from the Workflow Trace Archive, a collection of real-world workload traces in a standardized format [21]. The archive includes multi-day traces of real Alibaba and Google datacenters which researchers can use to evaluate new datacenter techniques in OpenDC.

2.2 TaskFlow: Energy-aware workflow scheduling

For this report, we have partially ported the TaskFlow algorithm to the OpenDC simulator. The algorithm exploits the assumption of knowing the runtime of a given task on a given machine by calculating a so-called *slack* value for that given task. The slack of a task is the amount of time it can "slack off" without delaying other tasks further in the workflow dependency graph. For example, in Figure 1 task *4* has two dependencies, task *1*, which has a runtime of 10 and task *2*, which has a runtime of 15. If task *1* took 15 instead of 10 time units, task *4* would not be delayed as it is limited by the runtime of task *2*.

Having established that certain workflow tasks can theoretically be slowed down without a total system runtime increase, the TaskFlow paper then defines two methods for slowing tasks down that increase energy efficiency. These two methods are *DVFS* (Dynamic Voltage and Frequency Scaling) and *heterogeneity*, of which this report only implements the heterogeneity method. The first method DVFS is a feature of modern processors that allows the programmer to select a clock speed and power budget for the processor. With lower clock speeds comes lower power consumption, and thus by running a task with slack on a slower and less power-hungry DVFS profile, the system's energy efficiency can be increased. The second method that TaskFlow employs is heterogeneity-aware scheduling. The machines in a heterogeneous datacenter have differing performance and energy characteristics. By scheduling tasks with slack on slower but more energy-efficient machines, the scheduler can improve the overall power efficiency of the datacenter.

The limitation of these two methods is that the available combinations of slowdown and increased energy efficiency are finite and defined by the hardware. The strength of the heterogeneity method is limited by the heterogeneity-factor machines present in the datacenter, and the strength of the DVFS method is limited by the number of "profiles" (called P-states by Intel) exposed by the processors. Slowdowns can occur if the available methods are insufficient for the amount of slack in the workflow.

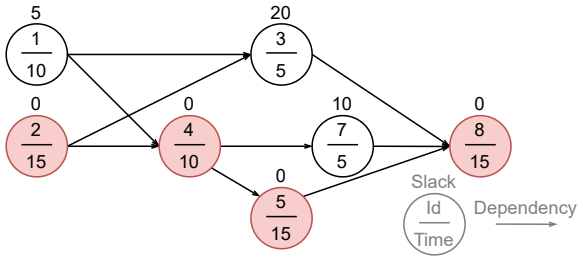

Figure 1: An example of a workflow annotated with slack for TaskFlow. Vertices highlighted in red are part of the critical path. Image is adapted from [20].

3 DESIGN & IMPLEMENTATION OF TASKFLOW IN OPENDC

This section will go into detail on our TaskFlow implementation in OpenDC. This is divided into two subsections: first, the slack calculation of tasks in an incoming workflow, and second the compute scheduler that exploits heterogeneity to allow tasks to "slack off" and improve energy efficiency.

3.1 Slack calculation

In the original implementation of the TaskFlow simulator, the slack was calculated offline using Spark for the Alibaba trace for the sake of speed. For a more realistic result, we implemented slack calculation in real-time in the OpenDC simulator. The slack is calculated for workflow tasks, as seen in Figure 1.

The algorithm to compute the slack of tasks can be broken down into three components: (1) the topological sorting of the workflow's DAG, (2) computing the earliest possible starting time of each task in the DAG using the topologically sorted DAG and (3) computing the slack based on the earliest possible starting time of a task's children. The complexity of each of those three operations is $O(|V| + |E|)$, leading to the complexity of the complete slack calculation being $O(3 * (|V| + |E|))$. For a complete description of the slack calculation algorithm, please refer to the TaskFlow paper[20].

3.2 TaskFlow compute scheduler

Laurens Versluis originally implemented the TaskFlow scheduler policy in *WTASim* [20]. WTASim supports the same Traces from the Workflow Trace Archive that OpenDC does, but it is more limited. Some functionality present in OpenDC was not included to make the simulator faster at a specific task.

As the simulators have different architectures, porting the policy from WTASim to OpenDC is not trivial. In WTASim, the policy can access almost all information, like how many FLOPs a task requires and what machines are available to provision on. In OpenDC, the simulator is split into modules. While this might make maintaining the simulator in the long term easier, it makes implementing the policy more cumbersome as not all information can be read from any module.

We will only be implementing the heterogeneity method that TaskFlow uses to improve energy efficiency. The OpenDC simulator

already supports heterogeneous datacenters, but it lacks the notion of a machine's "power-efficiency" that the TaskFlow scheduler needs to determine which machine to schedule a task on. WTASim defines power efficiency as TDP $*$ normalizedSpeed. This characteristic can be computed when all machines are loaded into the simulator.

The policy will be implemented as a compute scheduler which is part of the *opendc-compute* module. To determine what host a task will be provisioned, we need information about the slack from *opendc-workflow*, the specifications and power information of hosts from *opendc-simulator*, and of course, how many FLOPs a task requires to complete. This posed a cyclic dependency problem, where the scheduler needs information that goes against the flow of the dependency graph in OpenDC. This is visualized in Figure 2. We have looked at two ways to pass the required information to the scheduler, but they both rely on information only present at runtime making implementation more error-prone.

3.2.1 Design alternative: central repository.
The first design uses a central repository to store the slack information for tasks and power data for the hosts. By opting for this approach, a new repository needs to be created. This repository can either be a database or a simple key-value store accessible to all parts of the OpenDC code base. The existing code base needs to be extended in multiple places to store and retrieve data from the repository.

The central repository approach will consume the most memory because we are not just storing a copy of the data but also to which object it belongs. Doing frequent lookups from the repository also comes with additional overhead, which makes this option unappealing.

3.2.2 Design alternative: reflection.
The second alternative we will consider is using Java/Kotlin reflection to access fields of the objects that are only populated at runtime. Reflection allows programmers to modify the behaviour and state of objects when the program is running, which has been known to lead to error-prone code.

When we retrieve a field from the Task object, we would ideally like to cast it to its native type. Unfortunately, this is not possible as this would introduce a new dependency on another module. Instead, we can choose only to use primitive data types that are built into Kotlin, like integers, doubles, and strings. This would also require modifications to the code base in multiple places.

3.2.3 Final design: metadata.
The final design option used to implement TaskFlow relies on attaching metadata to objects. Metadata is modelled as a mutable dictionary with a string as key. This string is defined as a constant in all the modules where it is required so that it is accessible from the compute, simulation, and workflow modules. Similarly to the approach with reflection, we can not store complex objects and cast them as the classes fall out of scope. Instead of duplicating the classes and interfaces, we will only store primitive types. For slack, doubles suffice. Similarly, a combination of integers and doubles is sufficient for the other data that needs to be passed around.

As all objects in OpenDC already have a mutable dictionary, this approach requires little modifications to the existing code base. This makes it more appealing compared to the other two more complex alternatives. However, all three designs require code to be written in ways that go against common software-engineering practices as they all actively circumvent the architecture of the software stack. With those considerations, we implement this third and final design option.

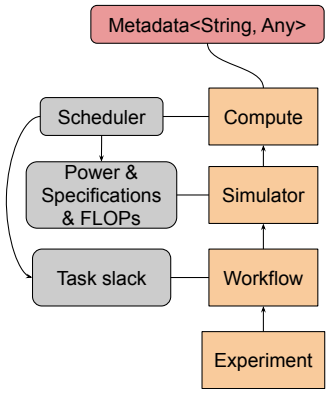

Figure 2: Diagram of the final design implementation of TaskFlow in OpenDC. The grey boxes are information that resides in the modules shown as orange boxes. The red box is the final implementation design from 3.2.2. The lines with arrows indicate dependencies, and the lines without arrows indicate ownership.

3.2.4 Implementation of the scheduler.
When allocating a new virtual machine (VM) on a host, the TaskFlow scheduler first filters the list of hosts to include only nodes with sufficient resources (RAM, CPU). It then sorts the list of available hosts by their power efficiency. A more efficient machine can do more FLOPs with less power and, since the list is sorted, has a higher chance of being allocated the VM.

When iterating through the filtered and sorted list of hosts, the slack stored in the metadata of a VM is used to determine if a task finishes in time on the selected host. This is possible because TaskFlow assumes the run time or at least the number of operations of a task is known. In our implementation, the normalized speed of all machines is used to determine a baseline execution time. For a selected host, the expected time on that host can then be calculated by dividing the baseline execution time by the normalized speed of that machine. The first machine that can execute the task within the original runtime plus the slack gets chosen.

Dynamic voltage and frequency scaling can clock down the allocated cores to reduce power consumption as long as the slack allows. This is, however, not implemented in this version, so it is expected to require a heterogeneous environment to perform well.

4 EXPERIMENTS

4.1 Monitoring metrics

We monitor the metrics generated during runtime using an OpenDC *monitor* object. This monitor accesses the metrics collected from the trace of each simulated host, server, and service. The value of each metric is updated every time the scheduler is executed,

which occurs every 100 milliseconds in our experiment (default for OpenDC).

Using the monitor, we measure the host-level metrics: CPU utilization (%), energy usage (Joules), and active tasks throughout the simulation. Together, these metrics show the effect of active tasks on CPU utilization and energy consumption for different topologies and schedulers.

While we focus in this research on energy consumption, we do wish to note that the simulator can report more information than we extract for this research.

4.2 Setup

In preparation for the running of the experiments, we first need to walk through the following steps to ensure we get meaningful results from the simulator.

First, we need to select the trace that the simulator will simulate. We found the following three traces that are compatible with our modified version of OpenDC. "Askalon ee", this trace is an engineering trace (the domain of the trace is engineering) and contains 3500 workflows making it the largest trace supported by our version of OpenDC. Furthermore, two small scientific traces can be chosen: "Pegasus P1" and "Pegasus P7". They contain one and 35 workflows, respectively, making them too small to be sufficiently insightful. It would have been insightful to run larger traces (e.g. the Alibaba trace with 4.2 million workflows), but these could not be run due to formatting errors and time constraints.

The second decision that must be made is to decide the structure of the datacenter itself. These are the properties of the various clusters and machines used in the experiment. The main difference is if a system is homogeneous (consisting of only the same kind of machine) or heterogeneous (consisting of machines with varying attributes such as CPU speed, the amount of memory available and its power usage).

The third step is to select the scheduler for the trace. In this paper, three schedulers are compared against one another. The first scheduler that is used is the naive scheduler. The naive scheduler of OpenDC is a reimplementation of the real-world nova-scheduler developed by OpenStack[1]. The second scheduler is the random scheduler, which compiles a list of hosts that meet the task's criteria (enough memory, CPU power, etc.) and then randomly selects a host that fits the criteria to schedule the task on. The final scheduler compared against is the TaskFlow scheduler, which is discussed in more detail in section 3.

Only the libraries already present in OpenDC are needed for running the experiments themselves. For the visualisation, three libraries in Python are utilised. These libraries are Matplotlib for the actual visualisation, NumPy to deal with sizable results from the experiment (for the statistics such as energy consumption) and finally, to convert the various data types (integer to float, for example).

4.3 Execution

As the OpenDC simulator is deterministic, and the experiments do not introduce randomness (the random scheduler uses a predefined seed), the experiments are run once. This also means that reproducing the experiments will result in the same results on the condition that random faults are not introduced (which is not the case in this report). Table 1 shows the experiments performed in this research.

In the table, various characteristics of the experiments can be observed. First, the experiment defines the type of experiment that has been conducted. Workload describes the trace used for the experiment, and the system cluster shows whether the cluster was heterogeneous or not (consisting of CPUs with varying clock speeds). And finally, it shows the scheduling policy/policies used in a particular experiment.

4.4 Goals

These experiments aim to determine the energy efficiency of provisioning policies used in cloud computing environments running workflows. It should allow for educated decisions regarding the compared provisioning policies that may reduce power consumption whilst maintaining acceptable performance. As power prices and interest in the environment have been increasing, this can be used to alleviate real-world problems and contribute to society.

4.5 Metrics

The main metrics that are of interest regarding the goal of this research are both the energy consumption and the overall utilization of the system. The way energy consumption is defined in this research is in Wh (Watt hours). This represents 3600J of energy used by the system and is used in the field when measuring efficiency [22]. On the other hand, the utilization of the system is measured in the percentage of the processing power available to the CPUs.

4.6 Parameters

The various parameters defined in Table 1 are now discussed in more detail. As discussed, this research compared two kinds of traces against one another. Namely, an engineering trace containing 3500 workflows and the Pegasus scientific workflows represent less intensive tasks as they have only a single workflow for P1 and 35 workflows for P7.

In Table 1, we use homogeneous and heterogeneous parameters. These imply that both systems have different topologies, which is the case. To make a fair comparison, we assign both topologies the same number of cores. This should give both the same amount of processing power and allow us to observe the impact of only changing the topologies. Table 2 shows the specifications of the homogeneous system. This system consists of a single type of host with 16 cores available and a TDP of 300. Table 3 shows the specifications of the heterogeneous system. The main difference between this topology and the homogeneous one is that it has hosts with varying cores and TDP performance available (representing processing power). This allows the scheduler to schedule tasks that are not immediately required on slower machines to potentially increase energy efficiency.

[1] https://docs.openstack.org/nova/latest/admin/scheduling.html

Table 1: An overview of the experiments. The bold text indicates the variables that differ from the standard configuration.

Experiment	Workload	System cluster	Scheduling policies
1. Policy	Askalon ee	Heterogeneous	**All**
2. Cluster structure	Askalon ee	**Hetero- and Homogeneous**	Taskflow
3. Workload	**Askalon ee, Pegasus P1 & Pegasus P7**	Heterogeneous	Taskflow

Table 2: The specifications of the homogeneous system

Cores	Speed (GHz)	# of hosts	Mem. per host (GB)	Core # per host	Host TDP
112	3.2	7	128	16	300

Table 3: The specifications of the heterogeneous system

Cores	Speed (GHz)	# of hosts	Mem. per host (GB)	Core# per host	Host TDP
32	3.2	1	256	32	480
48	2.93	6	64	8	300
32	3.2	2	128	16	100

4.7 Factors

Combining and summarising the previous sections, we can define what we want to discover using this setup. First and foremost, we want to observe the impact of the provisioning on the energy consumption of the simulated datacenter. The second area of focus is the impact of homogeneity and heterogeneity on the simulated datacenter's performance (mainly the energy consumption) and observing if they differ. Finally, the following section shows the results of our findings.

5 RESULTS

For the first experiment, the result of running the TaskFlow policy with the Askalon ee trace is shown in Figure 3. This figure contains graphs of energy usage, CPU utilization, and active tasks. Plotted with the green line, we see the number of active tasks running on the system. This is the number of workflows that TaskFlow has scheduled and is running at the given scheduler cycle. The figure shows that more scheduled workflows lead to higher CPU utilization. The power usage fluctuates between 18 to 25 Wh per cycle. It seems to increase when the number of active tasks and CPU utilization rises. However, when there are no active tasks and the system is idle, the power consumption does not go any lower than 18 Wh per cycle. From the figure, it looks like a straggler task causes the other tasks at around cycle 2600 to wait.

To compare the energy consumption of the three policies, we look at the energy usage distribution in the violin plot depicted in Figure 4. The results for the random policy stand out as the largest part of the distribution is on the right side of the plot, which means the policy almost consumes a consistent amount of energy during the run. The plot shows a bimodal distribution for both the TaskFlow and naive scheduler. When comparing this to the violin plot with the CPU utilization, shown in Figure 5, this seems to be caused by the system either being idle with very low CPU utilization or loaded with a very high CPU utilization. Looking at Table 4, TaskFlow consumes more energy than the naive policy as it has a longer runtime.

The random scheduler performs notably worse than the other two schedulers that are compared for experiment one. The power usage, CPU utilization, and runtime duration are higher. The number of active tasks for the random scheduler, which is not depicted in a figure due to space constraints, is also two orders of magnitude higher than the number of active tasks for the other schedulers.

Table 4: The total energy usage and uptime for a naive, random, and TaskFlow scheduler.

Scheduler	Total energy usage (Wh)	Uptime (s)
Naive	60485.61	7.91e+08
Random	45713.68	4.86e+08
TaskFlow	68267.94	9.10e+08

For the second experiment, the TaskFlow policy running the Askalon ee trace is run on two topologies, and the energy distribution is plotted in Figure 6. These results show that the average energy usage of the homogeneous topology is lower than the average energy consumption of the heterogeneous topology. However, looking at Table 5 shows that the homogeneous topology takes longer to execute the same trace and consumes more power overall.

Table 5: The total energy usage and uptime for a homogeneous and heterogeneous topology.

Topology	Total energy usage (Wh)	Uptime (s)
Homogeneous	111798.00	15.07e+08
Heterogeneous	68267.94	9.10e+08

The third experiment compares traces of different workload types in Table 6. We compare the scientific workload of Pegasus P1 [23] and Pegasus P7 [24] to the engineering workload from the Askalon ee trace. The table shows that the Pegasus P1 trace has a significantly shorter uptime than the other traces, which is reflected in its energy usage, as seen in appendix Figures 7 and 8. The scientific workload from the Pegasus P7 trace also differs from Askalon ee's engineering workload.

6 DISCUSSION

The first experiment shows that the selection of a provisioning policy greatly impacts the power consumption of the system. TaskFlow uses less energy per cycle on average but takes longer to

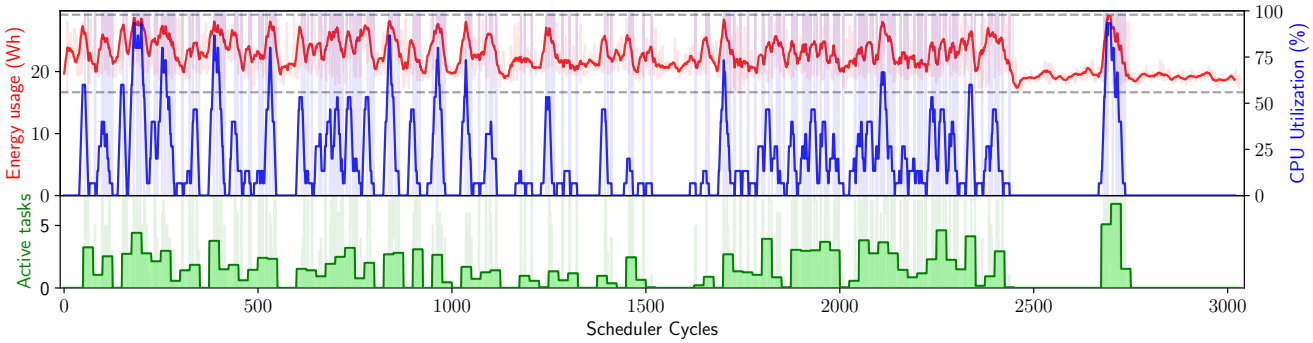

Figure 3: Workflow energy usage compared to CPU utilization over time using a rolling average of 15. The number of active tasks admitted by the scheduler is depicted in green. The TaskFlow scheduler was used on the Askalon ee trace.

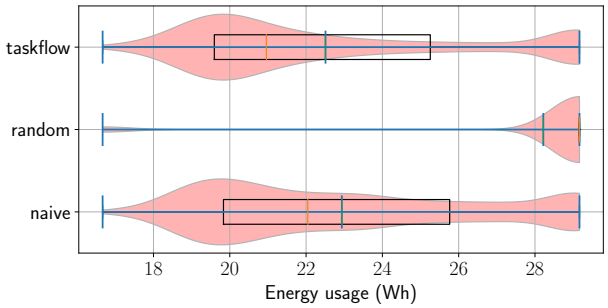

Figure 4: Distribution of energy consumption for different schedulers.

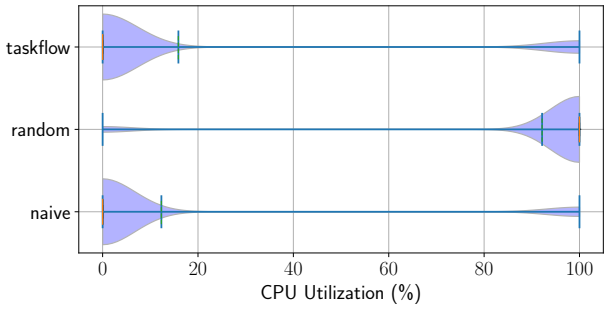

Figure 5: Distribution of CPU utilization for different schedulers.

complete the workflow than the naive scheduler inspired by the nova scheduler from OpenStack. We suspect this may be caused by a straggler process that prevents other tasks from being executed as the CPU utilization is low around that time. We theorize it may also be caused by tasks being postponed when no resources were available and not being rescheduled immediately as they became available.

The power usage of nodes in idle does not drop down to zero. This is caused by the nodes not being shut down and processors not clocked back when there are no tasks to run. Had dynamic voltage and frequency scaling (DVFS) been implemented, the idle power

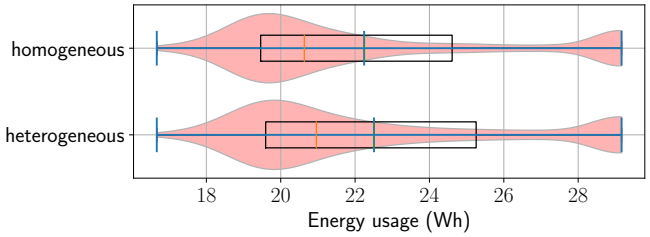

Figure 6: Distribution of energy consumption for a heterogeneous and homogeneous topology.

Table 6: The total energy usage and uptime for the Askalon ee, Pegasus P1, and Pegasus P7 traces. The 'engineering' and 'scientific' workload types are abbreviated to 'Eng.' and 'Sci.', respectively.

Trace	Type	Total energy usage (Wh)	Uptime (s)
Askalon ee	Eng.	68267.94	9.10e+08
Pegasus P1	Sci.	3660.46	0.57e+08
Pegasus P7	Sci.	73941.81	9.80e+08

draw could have been reduced. In addition, TaskFlow could have used DVFS to get the most out of the slack by downclocking the core on which a task is being run. In the original work of Versluis, DVFS was implemented, and the results were more positive.

The random policy was included to show that any scheduler is better than none. The results show that the random scheduler admits too many workflows at once. This overprovisioning overloads the system, as shown in the plots with the CPU utilization and power usage. When the system is overprovisioned, context switching overhead reduces the performance, and the runtime of the simulation is increased.

From the second experiment, we conclude that our implementation of TaskFlow performs better with a heterogeneous topology than with a homogeneous topology. As our implementation does not implement DVFS, there is no difference in efficiency between nodes. This efficiency is used to sort and select a node, which limits the TaskFlow policy's functionality.

To compare the effectivity of TaskFlow across different workflows, we used experiment three. The results show that the execution time of Pegasus P1 is shorter than Pegasus P7. This was to be expected as the former is shorter than the latter. It would have been better to vary more than the workload in this experiment, as there is too little data to give any meaningful conclusion about the scheduler's effectiveness across multiple workload types.

As starting with OpenDC and porting the TaskFlow policy to OpenDC was not trivial, we have included a small description of running an end-to-end experiment in Appendix A. This describes the process of defining an experiment, including the inputs and scheduling policies and how to run it. As our implementation and plotting scripts are open-source, these too can be taken as inspiration in case this work is to be repeated or extended.

7 THREATS TO VALIDITY

The OpenDC simulator uses assumptions regarding the FLOP workload of tasks. From the traces, the actual runtime is multiplied by a magic number to determine the number of FLOPs. This magic number is a constant that does not change depending on the trace. This is a strong assumption, as it is unlikely that all traces have been run on the same hardware. Another factor that the simulation does not consider is outside factors like weather conditions (heatwaves) that could impact the data centre's performance and energy consumption. All the assumptions the simulator makes or factors it is (currently) unable to simulate increase the reality gap between the simulation and the real world.

Another limit is how topologies are read from a configuration text file and used in the simulation module of the simulator. This configuration file contains information about the number of cores and the processor's clock frequency, among other things. These options do not enable the user to quickly model the generational improvements in instructions per cycle (IPC) to compare newer hardware to old nodes. During our experiment, we assumed that all processors used in the nodes provided in the `heterogeneous.txt` and `homogeneous.txt` have an IPC of one.

8 CHALLENGES

We encountered multiple challenges during the design and implementation of the scheduler and the experiments. The first has to do with the complexity of the OpenDC simulator and the wide variety of experiments already implemented with it. This made it challenging to orchestrate the experiments' different components (like metric monitors and policies).

Furthermore, when creating the chosen experiments, the functionalities that were already implemented, like monitors, were not interchangeable between experiments. So they had to be implemented/changed. The process was further complicated because there was limited documentation available for OpenDC. In combination with the structure of the system, as shown in Figure 2, it made expanding the code with new features difficult.

9 RELATED WORK

This related work section will look at energy-aware research concerning scheduling and datacenters. For work relating to the TaskFlow algorithm, the original paper can be referenced [20].

9.1 Alternative approaches for energy-aware scheduling in datacenters

Instead of manually developing energy-aware new policies, machine learning can be used to improve scheduling decisions [25]. In this work, a machine learning algorithm is trained with a labelled dataset consisting of Grid workloads. Their findings show improvement in utilization and power consumption compared to standard round-robin scheduling. However, as the workload is not the same as the workloads tested in our work, we can not directly compare TaskFlow to this trained algorithm.

The most powerful method to decrease energy usage is simply turning machines off. However, in the era of virtual machines and live migrations, there has been work that tries to optimize for the lowest number of powered-on machines while still adhering to service level agreements [26].

9.2 Energy-aware process scheduling in single-node systems

This report focuses on the energy-aware scheduling of a datacenter, e.g. a cluster of many compute nodes. There has also been a significant research effort on the scheduling issues inside a single-node system such as a Linux machine [27]. As some of these systems are battery-operated and not bound by service level agreements, these schedulers can be more aggressive regarding frequency scaling. Notable similar works to TaskFlow include the GRUB-PA algorithm currently implemented in Linux that integrates the frequency-scaler and CPU scheduler to improve energy-efficiency on ARM cores [28], and the HASS algorithm that is heterogeneity-aware on multi-core systems [29].

9.3 Energy scheduling in datacenters

In case the cost factor of energy is the most significant motivator for reducing energy usage, one could also optimize how a given amount of energy is utilized, as opposed to lowering the amount of energy consumed. For example, work has been done in the direction of employing stored energy solutions to reduce energy costs [30–32].

With the advent of cheap green energy, it can also be advantageous to schedule workloads when this specific type of energy is plentiful (e.g. when the sun is shining and the wind is blowing) [33, 34].

10 CONCLUSION

This report demonstrates the impact of the TaskFlow scheduler on the energy consumption and resource utilization of three workloads originating from both the scientific and engineering community. By comparing the TaskFlow scheduler to a naive and random scheduler, we have shown that TaskFlow reduces energy consumption at the cost of scheduler cycles. However, this reduction in energy consumption only holds for heterogeneous clusters containing efficient nodes, as dynamic voltage and frequency scaling (DVFS) is not included in the current version. DVFS allows for additional energy-saving strategies, as is shown by [20]. These findings show that TaskFlow can effectively reduce energy consumption in simulated datacenters, but further investigation into its capabilities and limitations is required.

10.1 Future Work

The TaskFlow policy is just one manually created policy, but recent work shows that machine learning algorithms trained on a labelled data set can also take good scheduling decisions regarding energy consumption and performance [25]. Furthermore, by defining a value function based on metrics deemed necessary, this algorithm can be trained to operate under various conditions. As it is possible to extend the OpenDC simulator, interesting future work could include experiments with machine learning to create scheduling policies and compare them to handcrafted policies.

It would also be interesting to run and compare more and larger traces, like the Alibaba cloud trace from the Workflow Trace Archive [35]. The Alibaba trace, in particular, is much larger than the traces tested for this report and might contain longer dependency chains and more scenarios where using slack can reduce energy consumption.

During the experiments, it was noticed that not all traces from the AtLarge research trace archive follow the predefined format. Some traces do not work out-of-the-box with the OpenDC simulator when opened with the workflow loading method of the *opendc-workflow-service* module, partly due to misformatted traces. To aid future research, these traces could be validated, cleaned and tested so that they are confirmed to work with the simulator without additional modification.

ACKNOWLEDGMENTS

We thank Alexandru Iosup for his guidance during the project period, especially during the weekly meetings and presentation sessions. We would also like to thank Sacheendra Talluri for setting us on the right path to understanding OpenDC. And lastly, Fabian Mastenbroek for helping when bugs were discovered in OpenDC or the data.

REFERENCES

[1] A. S. G. Andrae and T. Edler, "On global electricity usage of communication technology: Trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015. [Online]. Available: https://www.mdpi.com/2078-1547/6/1/117

[2] W. Deng, F. Liu, H. Jin, B. Li, and D. Li, "Harnessing renewable energy in cloud datacenters: opportunities and challenges," *IEEE Network*, vol. 28, no. 1, pp. 48–55, 2014.

[3] C. B. voor de Statistiek, "Inflation rate down to 9.9 percent in november," Dec 2022. [Online]. Available: https://www.cbs.nl/en-gb/news/2022/49/inflation-rate-down-to-9-9-percent-in-november

[4] R. Koster, "Bedrijven in de knel door extreem hoge energieprijs," *NOS*, Oct 2021. [Online]. Available: https://nos.nl/collectie/13880/artikel/2400080-bedrijven-in-de-knel-door-extreem-hoge-energieprijs

[5] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 338–347. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/1815961.1816004

[6] J. Zhao, J. Liu, H. Wang, C. Xu, W. Gong, and C. Xu, "Measurement, analysis, and enhancement of multipath tcp energy efficiency for datacenters," *IEEE/ACM Trans. Netw.*, vol. 28, no. 1, p. 57–70, feb 2020. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1109/TNET.2019.2950908

[7] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "Gpuwattch: Enabling energy optimizations in gpgpus," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 487–498. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/2485922.2485964

[8] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 47–56. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/2145694.2145704

[9] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-efficient cnn implementation on a deeply pipelined fpga cluster," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ser. ISLPED '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 326–331. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/2934583.2934644

[10] I. n. Goiri, T. D. Nguyen, and R. Bianchini, "Coolair: Temperature- and variation-aware management for free-cooled datacenters," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 253–265. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/2694344.2694378

[11] Q. Pei, S. Chen, Q. Zhang, X. Zhu, F. Liu, Z. Jia, Y. Wang, and Y. Yuan, "Cooledge: Hotspot-relievable warm water cooling for energy-efficient edge datacenters," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 814–829. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/3503222.3507713

[12] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee, "Task-aware virtual machine scheduling for i/o performance." in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 101–110. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/1508293.1508308

[13] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Workshop on job scheduling strategies for parallel processing*. Springer, 2003, pp. 44–60.

[14] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 69–84. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/2517349.2522716

[15] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," *SIGPLAN Not.*, vol. 48, no. 4, p. 77–88, mar 2013. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/2499368.2451125

[16] A. Iosup, G. Andreadis, V. Van Beek, M. Bijman, E. Van Eyk, M. Neacsu, L. Overweel, S. Talluri, L. Versluis, and M. Visser, "The opendc vision: Towards collaborative datacenter simulation and exploration for everybody," in *2017 16th International Symposium on Parallel and Distributed Computing (ISPDC)*, 2017, pp. 85–94.

[17] F. Mastenbroek, G. Andreadis, S. Jounaid, W. Lai, J. Burley, J. Bosch, E. van Eyk, L. Versluis, V. van Beek, and A. Iosup, "Opendc 2.0: Convenient modeling and simulation of emerging technologies in cloud datacenters," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2021, pp. 455–464.

[18] G. Andreadis, F. Mastenbroek, V. van Beek, and A. Iosup, "Capelin: Data-driven compute capacity procurement for cloud datacenters using portfolios of scenarios," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 26–39, 2022.

[19] L. Versluis, M. Neacsu, and A. Iosup, "A trace-based performance study of autoscaling workloads of workflows in datacenters," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2018, pp. 223–232.

[20] L. Versluis and A. Iosup, "Taskflow: An energy- and makespan-aware task placement policy for workflow scheduling through delay management," in *ICPE '22: ACM/SPEC International Conference on Performance Engineering, Virtual Event, China, April 9-13, 2022, Companion Volume*, ACM, Ed. ACM, 2022. [Online]. Available: https://doi.org/10.1145/3491204.3527466

[21] L. Versluis, R. Mathá, S. Talluri, T. Hegeman, R. Prodan, E. Deelman, and A. Iosup, "The workflow trace archive: Open-access data from public and private computing infrastructures," *IEEE Trans. Parallel Distributed Syst.*, vol. 31, no. 9, pp. 2170–2184, 2020. [Online]. Available: https://doi.org/10.1109/TPDS.2020.2984821

[22] A. Guleria, J. Lakshmi, and C. Padala, "Quadd: Quantifying accelerator disaggregated datacenter efficiency," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019, pp. 349–357.

[23] R. Prodan and A. Iosup, "Pegasus_p1 trace from the atlarge workflow trace archive," Dec 2022. [Online]. Available: https://doi.org/10.5281/zenodo.3254588

[24] ——, "Pegasus_p7 trace from the atlarge workflow trace archive," Dec 2022. [Online]. Available: https://doi.org/10.5281/zenodo.3254544

[25] J. L. Berral, I. n. Goiri, R. Nou, F. Julià, J. Guitart, R. Gavaldà, and J. Torres, "Towards energy-aware scheduling in data centers using machine learning," in *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, ser. e-Energy '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 215–224. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/1791314.1791349

[26] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, ser. MGC '10. New York, NY, USA: Association for Computing Machinery, 2010.

[Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/1890799.1890803
[27] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM Trans. Embed. Comput. Syst.*, vol. 15, no. 1, jan 2016. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/2808231
[28] C. Scordino, L. Abeni, and J. Lelli, "Energy-aware real-time scheduling in the linux kernel," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, ser. SAC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 601–608. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/3167132.3167198
[29] D. Shelepov, J. C. Saez Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar, "Hass: A scheduler for heterogeneous multicore systems," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, p. 66–75, apr 2009. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/1531793.1531804
[30] D. Wang, C. Ren, A. Sivasubramaniam, B. Urgaonkar, and H. Fathy, "Energy storage in datacenters: What, where, and how much?" in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 187–198. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/2254756.2254780
[31] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar, "Benefits and limitations of tapping into stored energy for datacenters," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 341–352. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/2000064.2000105
[32] S. Govindan, D. Wang, A. Sivasubramaniam, and B. Urgaonkar, "Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVII. New York, NY, USA: Association for Computing Machinery, 2012, p. 75–86. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/2150976.2150985
[33] I. n. Goiri, K. Le, M. E. Haque, R. Beauchea, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenslot: Scheduling energy consumption in green datacenters," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/2063384.2063411
[34] F. Kong and X. Liu, "A survey on green-energy-aware power management for datacenters," *ACM Comput. Surv.*, vol. 47, no. 2, nov 2014. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/2642708
[35] A. 2018, "Workflow trace archive alibaba2018 trace," Jul. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3254606
[36] R. Prodan and A. Iosup, "Askalon_ee trace from the atlarge workflow trace archive," Dec 2022. [Online]. Available: https://doi.org/10.5281/zenodo.3254471

A END-TO-END OPENDC EXPERIMENT

OpenDC is a large and complex datacenter simulator which consists of many interconnected modules. An environment and its parameters need to be set properly in order to model a datacenter and execute simulations within OpenDC. Orchestration of an experiment is performed by creating a runner in the OpenDC environment. In the following sections a walkthrough of the experiment runner and its internal components is provided, and the visualisation process is briefly highlighted. In this walkthrough the parameters from the *Policy* experiments in this report are used to demonstrate the End-to-End OpenDC experiment process. The functionalities and source code used to perform the end-to-end experiment can be found in a public repository[2], which is a fork of the original OpenDC repository.

A.1 Input Parameters

Before an experiment can be executed, the corresponding inputs and parameters need to be defined. One of the major inputs for an experiment is the trace. A trace describes properties of the workflows and tasks that are to be executed. In our experiments the traces are from the AtLarge Research Workflow Trace Archive[3], which makes use of the WTF trace format within OpenDC. In this walkthrough we will make use of the askalon_ee trace[4][36], which is a trace from the engineering domain.

Besides the trace, the topology of the system also needs to be defined. The topology can be defined through a txt file. A topology txt file is structured using the following compute resources property descriptors:

(1) ClusterID
(2) ClusterName
(3) Cores
(4) Speed
(5) Memory
(6) numberOfHosts
(7) memoryCapacityPerHost
(8) coreCountPerHost

Each row in the file describes a set of hosts and their corresponding properties. Within OpenDC this file is converted into the corresponding hosts objects which will be used throughout the experiment.

Next, the scheduling policies need to be defined. The experiment allows the user to set 5 different policies at different levels in the system:

(1) JobAdmissionPolicy
(2) JobOrderPolicy
(3) TaskEligibilityPolicy
(4) TaskOrderPolicy
(5) allocationPolicy

Within OpenDC, workflows are denoted by Job objects. The JobAdmissionPolicy determines whether a workflow is admitted to a scheduling cycle. After the Jobs are admitted, the JobOrderPolicy dictates the order in which the workflows are placed in the scheduling queue. A task in the workflow is indicated by a Task object in OpenDC. The TaskEligibilityPolicy elects the tasks allowed in the scheduling cycle. Eligible tasks are ordered by the TaskOrderPolicy and provided to the scheduling cycle. Lastly, the tasks need to run on compute resources. The allocationPolicy defines how the compute hosts are selected for a task that needs to run.

In this walkthrough, both the JobAdmissionPolicy and the TaskEligibilityPolicy are set to always allow a workflow or task to be processed. Both the JobOrderPolicy and the TaskOrderPolicy order the workflow and tasks based on submission time. For the allocationPolicy the TaskFlow scheduler is used.

Listing 1: Scenario class

```
public data class Scenario(
    val topology: Topology,
    val workload: Workload,
    val schedQuantum: Duration,
    val jAdmissionPolicy: JobAdmissionPolicy,
    val jOrderPolicy: JobOrderPolicy,
    val tEligPolicy: TaskEligibilityPolicy,
    val tOrderPolicy: TaskOrderPolicy,
    val allocationPolicy: String,
    val opPhenomena: Phenomena
)
```

A.2 Experiment Runner

Once the input parameters are set. The experiment runner can start the experiment. The runner handles the setup of the internal components and monitors required to execute an end-to-end experiment. Instantiation of the runner also sets the resource's path (i.e., for a path to the folder containing data such as the topology description) and output file path. A *scenario construct* is used by the experiment runner to define and keep track of the experiment parameters, such as system topology and allocation policies. Listing 1 demonstrates the scenario construct.

It should be noted that, currently, the OpenDC workflowservice does not support the operationalPhenoma displayed in the Scenario construct. OperationalPhenoma will not be used and, therefore, experiments are deterministic.

When the runner has received the scenario parameters, it can start the setup of the environment. The OpenDC provisioner is arranged to define the configuration within OpenDC. In the provisioned, the computeservice, hosts, monitor, and workflowservice are registered and constructed. The computeservice houses the TaskFlow scheduler. Host objects are created and registered based on the description in the topology file. A monitor is registered to keep track of all statistics from the compute domain (CPU utilisation, energy usage, and more) during the execution of the experiment. Data is collected by the monitor every scheduling cycle (occurs at least every 100 milliseconds).

After everything has been set up, the workflowservice can replay the workload defined in the trace. It mimics the incoming workflows and tasks as if all the services would be running live in

[2]https://github.com/Peter-JanGootzen/opendc
[3]https://wta.atlarge-research.com/
[4]https://wta.atlarge-research.com/askalon_ee.html

production. The Job and Task policies are present and active in the workflowservice and the TaskFlow scheduler is active in the computeservice. When the replay has been completed, the results from the monitor are stored in a CSV file to be used in the visualisation and analysis stage.

A.3 Visualisation

The results from the experiment describe the activity and other aspects of the simulated datacenter and its scheduler. Data from the experiments are best expressed and analysed through visualisations. In order to create the desired visualisations, a Python script was created. This script parses the results file and processes the data. Finally, it can also combine data from separate experiments to visualise, for example, an energy consumption comparison between schedulers. The structure and functionality of such a visualisation script will highly vary depending on the needs of the experiments. The data from the experiment described here is shown in Figures 3 and 4.

B ADDITIONAL EXPERIMENT VISUALISATION

We visualize the energy consumption and CPU utilization of the Pegasus P1 and Pegasus P7 traces in Figures 7 and 8, respectively.

C A DISCUSSION ON A REVISED OPENDC ARCHITECTURE FOR ENERGY-AWARE SCHEDULING

In Section 3, the design and implementation considerations of TaskFlow in OpenDC were laid out. OpenDC employs a layered architecture in a statically typed language (Kotlin). The information required by an energy-aware scheduler crosses the boundaries that the architecture of OpenDC has set out. This led to a design and implementation that can be accurately described as "hacky". With the importance of energy awareness in our current-day society, it would be desirable that OpenDC allows researchers to employ energy-aware techniques in datacenter simulations.

Although we are not the most knowledgeable set of people on the current architecture of OpenDC and the considerations that went into it, we would like to suggest what we think would be the possible ways forward for OpenDC and energy-aware scheduling.

We propose two options that would achieve our objective: collapsing architectural layers coupled with a method to delegate scheduling to an experiment and a language change.

Our first proposed solution is to collapse several architectural layers of OpenDC, specifically the Compute and Simulator layers. This would alleviate the issue of retrieving FLOPs required for a VM execution completion, machine power information and machine specifications for its performance characteristics. Collapsing the Compute and Simulator layers would, however, not solve the issue of having access to the slack of a task in the scheduler. We think this can be solved by a new kind of scheduler. The current way to implement a scheduler in OpenDC is to implement a function in the Compute module that takes a set of machines available and a VM to schedule and returns which machine to schedule on. If this scheduling could be performed outside the Compute module and inside a higher-level module such as the Workflow module, then information about workflows (and thus slack) could be combined with the scheduling decision.

Our second proposed solution is to change the language into one of a looser type than Kotlin could also accomplish the goal of making OpenDC's functions more accessible. However, there are some major downsides to doing this as well. First, it would be a time-intensive task to port the entire project to another language. Secondly, there is always the chance that going from a static language to a more flexible one introduces bugs or could even cause the entire simulator to break down.

Lastly, it should be noted that it could also be an option not to change anything for energy-aware scheduling, as we were able to implement a state-of-the-art energy-aware scheduler successfully and our proposed options require significant development work.

D TIME LOG

Table 7 demonstrates the time spent throughout the lab project separated into categories.

Table 7: Estimate of the categorised time log.

Tag	Hours
think time	103
development time	163,5
experiment time	29
analysis time	27
write time	115
wasted time	26,5
Total	**464**

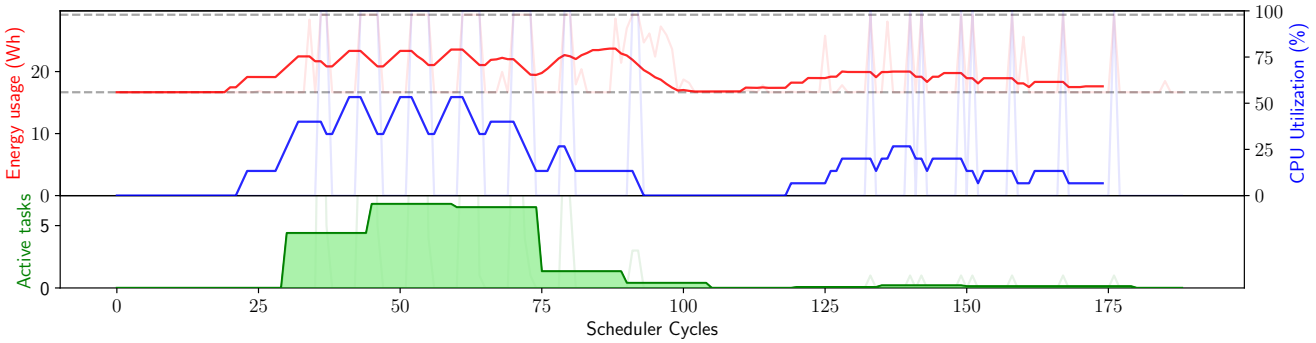

Figure 7: The Pegasus P1 trace's energy usage compared to CPU utilization over time using a rolling average of 15. The number of active tasks admitted by the scheduler is depicted in green. The active policy was TaskFlow.

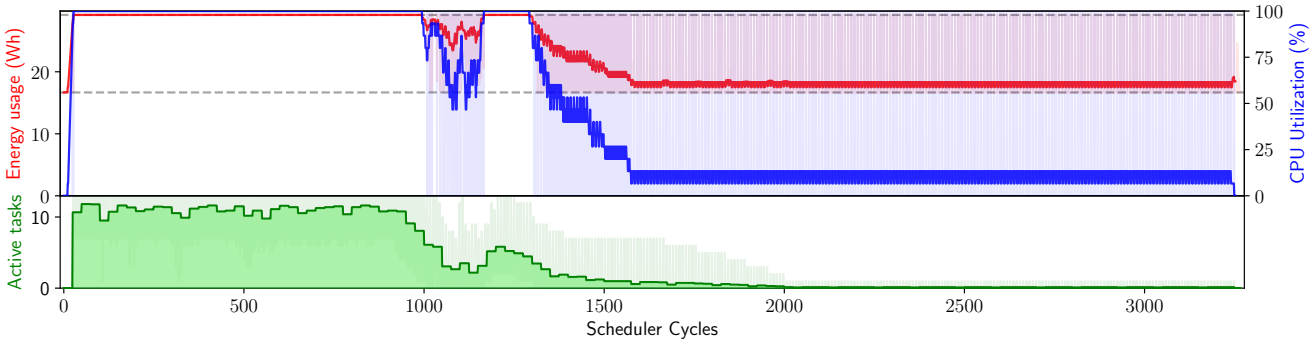

Figure 8: The Pegasus P7 trace's energy usage compared to CPU utilization over time using a rolling average of 15. The number of active tasks admitted by the scheduler is depicted in green. The active policy was TaskFlow.