A Carbon Aware Scheduler and Provisioner

Alexander Bridgwater† Axel Lundberg† Filip Skogh† ...
†Chalmers University of Technology
{ alebri, axelund, filipsk } @ chalmers.se

1 Introduction

As the improvements in power efficiency of hardware no longer can keep up with the exponential computational growth, today's researchers find themselves in a new position where efforts in reducing carbon emissions related to distributed data-centers needs a new approach. Various papers during recent years have suggested that more work needs to be focused on reducing carbon footprint of computations[C][C]. Especially when considering data center makes up $1-2\%$ of the world's carbon usage[C].

The intermittent nature of renewable energy and the lack of green large-scale energy storage makes it difficult for a data-center to fully rely on renewables. Therefore it is beneficial to move computations to data-centers with low carbon footprint. A carbon aware scheduler (CAS) can do this while controlling the amount of extra latency introduced contrary to traditional schedulers that only consider latency. A provision step can be added that moves the compute-resources to low-carbon regions using predictons, which we will refer to as a carbon aware provisioner (CAP).

Using a CAS to shift workload towards regions with low carbon intensity to reduce present emissions, also entails a positive second-order effect as it incentives new data centers to be built in areas with more renewable resources; given that these schedulers are applied as intended. Data-centers in polluted areas would be underutilized and thus unprofitable, having a direct impact on the amounts of brown energy produced in the region.

In general, compute-tasks can have two degrees of freedom: space and time. Compute intensive tasks e.g., ML and batch-processing can often utilize both. While time sensitive tasks such as web requests, have strict latency constraints that limit improvement possibilities in the time dimension. Instead the carbon intensity differential over space dictate the possible gains from moving computations across space.

1.1 Architecture

1.2 Assumptions

1. The properties of requests such as latency, load, and lifetime are uniform, hence requests are interchangeable. This simplifies the knapsack aspect of loading a server with partial loads, and produces more interpretable results, while staying realistic to some types of requests. This eases the constraints by refraining from considering properties on an application to application basis.

2. Requests considered are of type web requests. This puts latencies in the ballpark of between tens to hundreds nanoseconds. Note that this tightens the latency constrain compared to more extensive tasks such as model training in AI.

3. Complete knowledge of request rate for the incoming. By doing so, we can provide an implementation to showcase the potential of carbon loading without getting entangled in request rate predictions. Hence, we assume perfect predictions an hour ahead of all regions. As have been shown in "" by ..., this assumption is not far from reality.

4. We can start up computing units in any data-center. The big cloud providers are in principle never limited by the hardware resources at the data-centers[].

5. We assume instantaneous communication between regions and the scheduler. Although this disregards the distributed aspects of the system e.g. consistency, in this initial phase it allows us to focus on the technical aspect of scheduling. We consider this a solved problem for now and note that the additional latency for such a system is negligible.

6. We also limit the scope by ignoring the problem of capacity planning, i.e. setting the maximum servers, capacities, etc such that all demand can be satisfied.

2 Model

To formalize the scheduling and provisioning problems we device two mixed-integer quadratically constrained programs (MIQCP) as seen in Eq. 1-2. The server provisioner uses a carbon and request forecast for the next hour for the regions.

In the following problem definition all indices are in the set of regions, $\mathcal{R}$. We let $\bar{x} \in \{0, 1\}$ be a indicator variable for when no requests are sent to a region j, i.e. when $\sum_j x_{ij} = 0$.

2.1 Decision variables

x_{ij} := Requests from region i to region j
s_i := Number of servers in region i

2.2 Parameters

I_j := Carbon intensity in region j
λ_i := Request rate from region i
ℓ_{ij} := Latency from region i to j
c_j := Capacity for each server in region j
L_i := Maximum tolerated latency from region i
K := Maximum number of servers

2.3 Carbon Aware Provisioning Model

$$\min_{x} \quad \sum_{j} I_j \sum_{i} x_{ij} \tag{1a}$$

$$\text{s.t.} \quad \sum_{i} x_{ij} \leq s_j c_j \tag{1b}$$

$$\sum_{j} s_j \leq K \tag{1c}$$

$$x_{ij}\left(\ell_{ij} - L_i\right) \leq 0 \tag{1d}$$

$$\sum_{j} x_{ij} = \lambda_i \tag{1e}$$

$$\bar{x}_j s_j = 0 \tag{1f}$$

$$x_{ij}, s_j \in \mathbb{Z}_{\geq 0} \tag{1g}$$

2.4 CAS Model

$$\min_{x} \quad \sum_{j} I_j \sum_{i} x_{ij} \tag{2a}$$

$$\text{s.t.} \quad \sum_{i} x_{ij} \leq s_j c_j \tag{2b}$$

$$x_{ij}\left(\ell_{ij} - L_i\right) \leq 0 \tag{2c}$$

$$\sum_{j} x_{ij} = \lambda_i \tag{2d}$$

$$\bar{x}_j s_j = 0 \tag{2e}$$

$$x_{ij}, s_j \in \mathbb{Z}_{\geq 0} \tag{2f}$$

To solve the two MILPs we use the library `PuLP` which is a Python interface to the Coin-or branch and cut (CBC) solver. the relaxed MILP is solved quickly. To solve minimization problems 2 and 1 we relax them to MILP problems by removing the quadratic constraints 1f and 2e.

3 Simulation

The simulation aims to be simplistic and tries to capture the essence without expatiating about the level of abstraction the actual implementation of scheduler will be in. A central approach and a distributed approach have been considered, and the work has shifted between these two abstractions the last couple of weeks. The distributed approach considered is not a fully fledged simulation of a distributed system, but considers multiple scheduling algorithms deterministically. Figure 1 shows an overview each approach.

(a) Central scheduler (b) Distributed scheduler

Figure 1: Illustration of a central versus a distributed scheduler.

The scheduling scheme is updated every hour with the Carbon Aware Provisioning Model, see Section 2.3. Changing the scheduling scheme is equivalent to virtually "moving" computing units to different regions. Moving computing units to different regions is interchangeable to starting up computing units at one region and turning off computing units at another. As this is done only once every hour, the time it usually takes to start up computing units at another region is negligible.

To simulate this behavior in a simplistic manner, a "central" computing manager creates computing objects at different regions. The computing manager keeps track of regional data and the number of computing units as well as abstracting the delegation of requests to each computing unit.

3.1 Datasets

1. Carbon intensity. ElectricityMap [1] provides a diverse set of metric related to carbon for regions around the world. At this state of the project we shift the focus to four US-based regions during 2021. The regions can be seen in Table 1.

Table 1: Regions used in the simulation.

U.S Regions
California
Texas
Mid-Atlantic
Mid-West

The data metrics are depicted on an hourly basis. Since these carbon metrics are our primary decision variable (1b), the scheduling scheme is updated in hourly accordance to this data.

2. Request rate. We inject requests periodically over every hourly timestep to imitate realtime behaviour. Moreover, as the number of request tend to expand during the day and shrink during the night, the request load at a given time may differ substantially for closely located regions. Hence, we avoid using a static rate and base the rate of injections on a dataset supplied through our colleague containing request rates during 2011 on an hourly basis. Although request rates have surged in recent years, the daily distribution should persist.

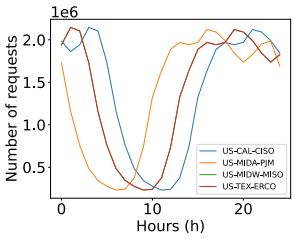
3.2 Testing

We use a test framework in Python called pytest [2]. Github allows us to use workflows to control how and when different types of actions should run depending on some event criteria. We have adopted a CI pipeline that automatically tests our code when commits are pushed to the main branch. This is very useful to identify breaking commits, and makes the development more test driven.

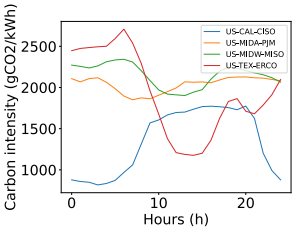
4 Results

The results are given for 1 day, namely the following date `2021-03-24`. Figure 2 shows the number of requests and the carbon intensity for each region during this day. We can see that the combined number of requests is peaks at timestep ~ 3 and ~ 20. Timestep 0 corresponds to `00:00` UTC. We can also

see that the regions with lowest carbon intensity are California and Texas while the highest are Mid West and Mid Atlantic. Texas has quite large carbon intensity between $0 - 8$. There is also a quite large differential between these regions, implying that there may be large gains to move computations from Mid West, Mid Atlantic and Texas to California.



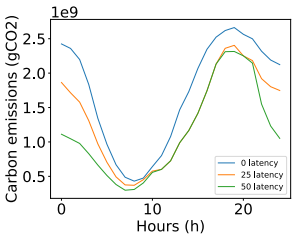
(a) Number of requests.



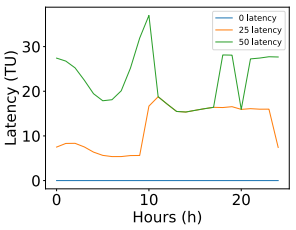
(b) Carbon intensity.

Figure 2: The figure shows the number of requests and carbon intensity for each region.

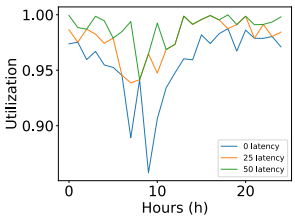
The simulation has been run three times with different latency constraints. One simulating infinite latency denoted as `50 latency`, one simulating no latency denoted as `0 latency`, and one that is a trade-off between latency and carbon intensity denoted as `25 latency`.



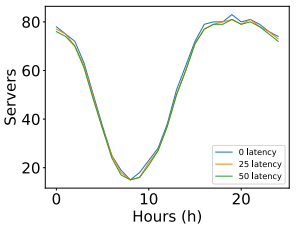
(a) Carbon emissions.



(b) Latency.

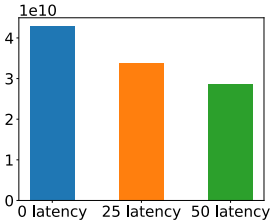


(c) Utilization (percentage).

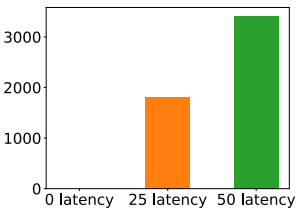


(d) Number of servers.

Figure 3: The figure shows each attribute with regards to a latency constraint of 0, 25 and 50.



(a) Carbon emissions.



(b) Latency.

Figure 4: The figure shows a comparison between carbon emissions and latency for each latency constraint.

5 Future Work

We have come to a state where the simulation makes sense under different circumstances. However, more analysis is needed to verify the correctness of the simulation. This includes reviewing the source code as well as performing data analysis on the outputs of the simulation. The current assumptions may be too limited and general. It could be useful to reflect if any changes are needed.

The latency between regions is calculated as the euclidean distance. Better results can be achieved by using a more accurate model for latency, especially if we are considering North America and Europe together.

It would be interesting to analyze our approach with more regions than 4. We currently have a dataset for Europe and a dataset for North America (with more regions), but with worse quality. The final work will contain more rigorous analysis for each of these datasets.

Finally, the assumptions should be inspected to be reduced where applicable in order to improve the accuracy of our results.

6 Conclusion

The results look promising. We have shown that scheduling requests and provisioning computing units across different regions successfully decreases total carbon emissions when there is a significant carbon intensity differential between regions. This supports previous research showing this differential lies around 30% for the US. However, more work is needed for a more thorough analysis.

7 References

[1] ElectricityMap. Electricitymap. `https://electricitymaps.com/`, 2022.

[2] pytest. pytest. `https://docs.pytest.org/en/7.1.x/`, 2022.