university of groningen / faculty of science and engineering

Comparing the carbon footprint of different cloud deployment models at BT Global Services

Masters Internship

2022 - v7

Student: A. Dijkstra

First supervisor: Dr. V. Andrikopoulos

Second assessor: C Bantis (BT)

External supervisor: R. Atherton (BT)

1 Introduction

Global emissions are increasing each year [IEA22]. The ICT share of emissions could be up to 3.9% of the total emissions [Fre+21]. Moreover, the prediction is this will only increase in the coming years [Fre+21]. Private and public cloud is a big part of this. AWS claims to have up to 93% reduction of emissions when changing from private to public cloud [Res19]. The first step in decreasing the carbon footprint is creating more insight into the data. Then educated choices can make public and private cloud greener.

BT Global hosts servers for their clients. These are mainly private cloud infrastructures. Westerhof created in a previous internship a dashboard to monitor the emissions of these private cloud configurations [Wes21]. However, BT Global also wants to know the emissions of public cloud infrastructures, in specific AWS. With this a comparison between public and private cloud infrastructures can be made. For this the public cloud infrastructure needs to match the computer power of private cloud infrastructure as close as possible.

The Carbon Cloud Footprint Toolkit (CCFT)[1] gives insight into running public cloud servers. It estimates the emissions made based on the present bills. However, this toolkit does not predict the emissions which the public cloud infrastructure will make in the future. With a prediction, an educated choice can be made beforehand. For that reason, during this internship I changed CCFT slightly and developed a wrapper around it. More specifically, I added two functionalities. First, the ability to get a similar AWS configuration, then give an estimation of emissions. To test and validate this model we used a use case from BT Global. In this report we discuss our proposal for a method that will predict public and private cloud emissions.

The report is structured as follows: In Section 2 we will look at the background. In Section 3 we will discuss the design in detail and Section 4 will explain the implementation. The results of the method will be presented in Section 5 and discussed in Section 6. We will look at further work in Section 7 and conclude in Section 8.

2 Background

2.1 Emission Scopes

To give insight into different emissions factors within a supply chain a three-scope, or 'Tiers', model is used [WRI04]. There are some models which try to give more insight into emissions factors by including for example four tiers [MHW08]. However, given this is outside the scope of this research we will stay at a three-scope model for supply chains. The three commonly defined scopes as defined by the World Resources Institute (WRI) and World Business Council for Sustainable Development (WBCSD) are as follows [WRI04]:
Scope 1: Direct Greenhouse Gas (GHG) emissions such as company facilities and cars.
Scope 2: Indirect GHG emissions such as electricity and heating.
Scope 3: Other indirect GHG emissions such as purchased goods.
Cloud emissions are most often included in Scope 3 emissions if it is public cloud [Myt20]. Thus, companies do not need to report the amount of emissions.

2.2 Multi Criteria Decision Making

In order to get the prediction of a public cloud configuration, first the private cloud configuration needs to be mapped towards it. This can be done with the use of Multi Criteria Decision Making (MCDM). MCDM has a short history, but its development is growing exponentially [ZTK14]. Due to this, there are a lot of different methods. Based on the paper by Le Sun et al [Sun+14] we identify four different main methods.

The first method is the analytic hierarchy process (AHP) [Saa90]. Through pairwise comparisons, the algorithm determines the different weights of the properties. That is, the amount of emissions is more important than the price. By pairwise comparison, the different instances are given a weight for each property. For example instance X is better in price than Y, but worse in the amount of emission. These weights are done based on a scale, which gives more insight than an arbitrary number. For each property, the property weight and instance weight are then multiplied together. This gives a total cost for each instance. The instance with the lowest cost is the best option.

[1] https://github.com/cloud-carbon-footprint/cloud-carbon-footprint

The second method is multi-attribute utility theory (MAUT). MAUT bases the cost on a property weight multiplied by a utility function. For each parameter a, possibly different, function is defined [GVB11]. This returns a value between zero and one. Here zero indicates it scores bad and one indicates it scores well on this property.

The third method is outranking methods [Roy68]. Fundamentally, it works as follows: if X is at least a certain amount of percentage better than Y, then X outranks Y. This gives a list of possible options.

The fourth big method is Simple Additive Weighting (SAW) [AMY10]. It is the most used and basic of the methods. It creates a ranking score by multiplying each weight by its respective property. This score is then used to determine the best option.

MCDM has also been used for decision-making for Cloud services. The method that stood out was proposed by Zeng et al [ZZZ09]. This consists of two steps, first based on the criteria the instances are filtered. Second, a cost function is used to determine the best fit.

2.3 Estimations

CCFT uses a pipeline with different formulas to calculate the emissions. The final emissions are then given in co_2e, which is the carbon dioxide equivalent. This gives the ability to indicate all greenhouse gases emitted. In this section, we briefly go through each formula.

2.3.1 Energy usage

The estimation of emissions can not be done directly. Before the emissions are estimated the energy consumption is calculated. First, the usage of the services is determined. After the amount that the service is used is known we can calculate the amount of energy consumption used. This is done based on the Etsy Cloud Jewels, as explained below. This gives the energy coefficients for cloud computing and cloud storage. Others were determined by the CCFT.

The estimation of emissions has multiple steps. The emission estimations use the amount of energy consumed. Different parts of the instance consume energy. Etsy already researched this and defined the Etsy Cloud Jewels [Som+20]. These are coefficients to determine the energy consumption of different parts of the system. Etsy, however, did not define all the coefficients needed. The makers of CCFT determined the unknown coefficients [Inc22]. The sum of these values is then the total energy consumption for that instance.

2.3.2 Power Usage Estimate

Power Usage Estimate (PUE) is the efficiency with which a data center runs [Bel+08]. A PUE of one is very efficient and the data center uses all the energy. A higher value means the data center is less efficient. By multiplying the energy consumption from the instance with the PUE we get the total energy consumption.

2.3.3 Carbon Estimates

From the total kWh used we can estimate the carbon estimates. The total energy consumption is multiplied by the Grid Emission Factors. The amount of co_2e each kWh emits for the grid [Inc22].

2.3.4 Embodied Emission

The last part of the emission comes from the production of the hardware used [Inc22]. Green Software has published the Software Carbon Intensity Standard for this. Embodied emissions take a major part of the total emissions of cloud computing [Gup+20][2]. These emissions happen at the start and end of the lifetime of the device. In this model an average lifetime of 4 years is used.

[2] As this talks about personal devices the exact number might be a little bit different

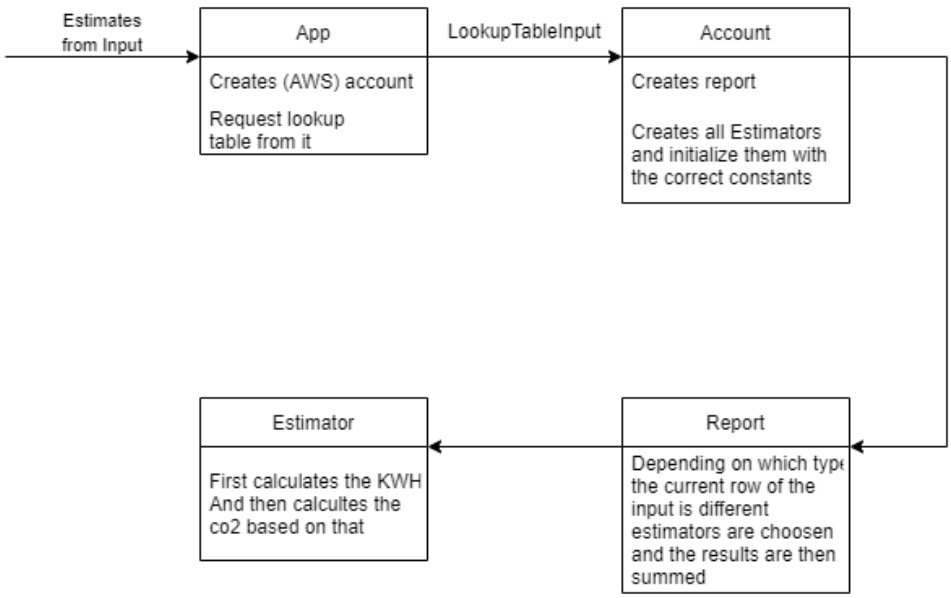

Figure 1: Scheme of the pipeline of the Cloud Carbon Footprint

3 Design

The CCFT can make an educated guess about the footprint of cloud computing. CCFT also has a module which creates a lookup table for all the services used. This gives for a certain usage unit the estimated amount of emissions. The different used types are hours, GB-Hours and GBs, where GB-Hours is the number of hours a certain amount of GB is stored.

An overview of the normal process for CCFT can be found in Figure 1. First, as input are the services in use. With this, it creates the account. In this example, we will use AWS services, so an AWS account is created. This forms the data in such a way the data can be used within the application. This is called the LookupTableInput.
Then the account creates a report class. This includes all the estimators with the correct constants.
Each estimator then calculates the emissions for each service in the account. This is done by first calculating the energy consumption in the same way as explained in Section 2.3.
The report class enumerates all the emissions and sums them.

The public cloud instance can be determined by the private cloud configuration. A wrapper will be created to translate the private cloud configuration towards a public cloud instance. At the start there is a list of around 500 AWS instances. With a filter the instances which cannot satisfy the requirements set by the private cloud configuration are removed. Then the fitter returns the best instance of those left based on user specified criteria. Those instances are then used as input for CCFT as explained by the previous paragraph. This gives an estimate for the emission based on the same structure.

CCFT also gives the possibility to determine on-premise emissions. For this the private cloud configurations only needed to be put into a format that was understandable for the CCFT model.

All the emissions from the AWS instances and the on-premise configurations will be put into a CSV file, which will function as the output report for the end-user. The file reports on the estimated energy consumption and emissions per instance.

4 Implementation

The wrapper is the part I designed and implemented. It consists of multiple stages. The pseudo-code can be found in Algorithm 1. As input, we take a list of all the configurations the private cloud has. Here is a

	On-Demand Windows pricing	On-Demand Linux pricing	Emission	vCPUs
On-Demand Windows pricing	1	5	4	7
On-Demand Linux pricing	0.2	1	0.5	3
Emission	0.25	2	1	3
vCPUs	0.14	0.33	0.33	1
positive/negative	-	-	-	+

Table 1: Pairwise weights Table used for AHP

very quick overview of the terms.

- **CPU**: Main processor.
- **RAM**: Random allocated Memory.
- **GPU**: Graphics card.
- **Architecture**: The instruction sets used (for example x64).
- **Threads**: The amount of threads per core.
- **NIC**: Network interface card, used to connect to the internet.
- **Storage**: The amount of storage a service has.
- **EBS**: Elastic Block Storage, a scalable solution from AWS to get more storage.
- **Bare Metal**: If a service is bare metal you only get the hardware. The public cloud vendor installed nothing else. This also means the hardware can directly be accessed.
- **DHS**: Dedicated Host Support, can you run everything on this server.

The pairwise weights, which is also the list of properties for the fitter, are given as input. The pairwise weights are used to determine the weights of the different properties. This is done the way Saaty proposed through an application of AHP [Saa90]. The pairwise weights are used as input instead of the property weights so the user has more control over the weights. An example of such weights can be found in Table 1. The final row is used for the utility function. This determines if it is positive or negative if there property has a higher value.

For each configuration, the algorithms finds the closest instance. This is added to the Lookup Table Input array. This is then used to create a look-up table.

A big part is how to find a service instance which is as similar to the original as possible. For this multiple criteria have to be used to make a decision, in other words MCDM. The four different methods which were taken into consideration can be found in the Section 2.2. The different methods were evaluated based on two major points. First, how easy is it for the end user to use it. Second, how good the reported results were. Since all four of the main methods had some problems, it was decided to use a combination of methods.

The pseudo-code to find the best instance can be found in Algorithm 2. The algorithm consists of two stages and is inspired by the method proposed by Zeng et al [ZZZ09]:
The filter first removes all instances which cannot be used. These are all the instances for which the configuration needs more/other resources than available.

Then the fitter decides which instance is the best fit. This part works as the MAUT methods mentioned in Section 2.2. For which an utility function determines the exact cost, except in my implementation we use a score function. There are two reasons why MAUT was chosen. Firstly a utility function was needed for some of the properties. Secondly, it made sure all the properties were evaluated the same way.

Algorithm 1 Wrapper

```
procedure PREDICTION(targetServerConfigurations, pairwiseMatrix)
    weights ← getWeights(pairwiseMatrix)
    i ← targetServerConfigurations.length − 1
    LUTInput ← []
    while i ≥ 0 do
        config ← targetServerConfigurations[i]
        instance ← privateToPublic(config, weights)
        LUTInput ← LUTInput + instance
        i ← i − 1
    end while
    LUT ← getLookUpTable(LUTInput)                ▷ Already defined in CCFT
end procedure
```

Algorithm 2 Private to Public cloud converter

```
procedure PRIVATETOPUBLIC(config, weights)
    instances ← allPublicCloudInstances
    instances ← filterInstances(instances, config)
    bestInstance ← instancesFitter(instances, config, weights)
    return bestInstance
end procedure
```

4.1 Filters

All instances are removed which do not have enough or the correct resources. The algorithm can be found in Algorithm 2. The following topics are used and filtered in the same order as below:

- CPU
  - Does it have enough vCpus
  - Does it have enough Cores
  - Is the clock speed high enough
- RAM, is there enough RAM
- GPU, two options:
  - No GPU needed, all instances with GPU's are removed
  - GPU's are needed, does the instance have enough GPU's
- Architecture, does the instance have the correct architecture.
- Threads, are there enough Threads
- NIC, are there enough Network Interface Cards
- Storage, is there enough storage.
- Bare Metal, two options:
  - Bare Metal is needed, then remove all others[3]
  - Bare Metal is not needed, then remove all bare metal instances.
- DHS, is a dedicated host needed.

[3]For BT Global this is a requirement, so this is always true

Algorithm 3 filterInstances

```
1: procedure FILTERINSTANCES(instances, config)
2:     i ← instances.length − 1
3:     while i ≥ 0 do
4:         if Filter is true then                ▷ This is done for all different filters
5:             Delete instances[i]
6:             Continue        ▷ Once removed, other filters doesn't need to be tested for this instance
7:         end if
8:         i ← i − 1
9:     end while
10:    return instances
11: end procedure
```

4.2 Fitters

Unfortunately, the filter does not remove all unwanted instances. To choose the best instance from the instances left 'Score' is introduced. Based on the requested fitter a score function is chosen. Within Algorithm 4 this is defined as getScore. This is dependent on the user's choice. Since multiple criteria are used it is very unlikely that two different instances have the same score. For that small likelihood there are instances with the same score we take the one which is the cheapest to rent.

The utility function 'getScore' is as follows and can be found in Algorithm 5. For the property, the minimum and maximum value are calculated beforehand by the function 'getPropertyDict'. These values can then be used by the utility function.

Algorithm 4 instanceFitter

```
1: procedure INSTANCEFITTER(instances, config, weights)
2:     i ← 0
3:     instanceScores ← ∅
4:     propertyDict ← getPropertyDict(instances, weights)
5:     minScore ← Infinity
6:     while i < instances.length do
7:         score ← getScore(instances[i], weights, propertyDict)
8:         instanceScores.push(instanceScores + score)
9:         minScore ← min(score, minScore)
10:        i ← i + 1
11:    end while

12:    fittedInstances ← ∅
13:    i ← instances.length − 1
14:    while i ≥ 0 do
15:        score ← instanceScores[i])
16:        if score == minScore then
17:            fittedInstances.push(instances[i])
18:        end if
19:        i ← i − 1
20:    end while
21:    return fittedInstances
22: end procedure
```

Algorithm 5 getScore

```
1: procedure GETSCORE(instance, weights, propertyDict)
2:     i ← 0
3:     score ← 0
4:     while i < weights.length do
5:         weight ← weights[i]
6:         currentVal ← instance[weight.key]          ▷ In other words iterate over all the property keys
7:         propMin ← properyDict[weight.key].min
8:         propMax ← properyDict[weight.key].max
9:         calcValue ← (currentVal − propMin)/(propMax − propMin)          ▷ This means value will be
   between 0 and 1
10:        if weight.config is negative then ▷ If the instance is better with a lower value on this property
11:            calcValue ← 1 − calcValue
12:        end if
13:        score ← score + (calcValue ∗ weight.value)
14:        i ← i + 1
15:    end while
16:    return score
17: end procedure
```

Filter name	Total removed
cpu	426
ram	0
gpu	7
arch	0
threads	0
nic	0
storage	27
metal	32
dhs	1
unknown	3
Total	496

Table 2: Amount of instances removed per filter

5 Case Study

This model was used on real-life data provided by BT Global for a specific client. The use case has four different servers running. This list was used as input. The results will be listed below and are split into the data and execution time.

5.1 Output

In total there are 504 possible instances from AWS at the start. The four private cloud configurations use the same architecture, except for the location. This meant the filter and fitter worked the same for all those four configurations. For that reason only the results of one configuration will be shown. The filter first removed most of the instances. The filters are used in the same order as seen in Table 2. For example, if it is already removed due to CPU considerations it is only counted towards that filter.

The fitter then chooses the best instance. This ended in one instance which was best suited for the tasks according to the model.

These instances were used as input for the AWS model and the initial configuration for the on-premise calculation. The program gives the emissions per instance. For simplicity's sake will we only look at the end result. This gives co_2e emission for one week of 46.21 Kg for the AWS instance and 140.86 Kg for the

# Instances	Sort	On-Premise	Create LUT	Estimate emissions	Create report	Total time
4	Min	6.503	189.357	4.638	0.405	238.9
	Avg	7.997	284.3358	6.8618	0.5235	343.3497
	Max	14.684	1793	30.331	0.847	1913
212	Min	23.13	7073	91.844	1.331	7272
	Avg	26.31895	7731.15	110.08075	1.60485	7958.1
	Max	36.657	8148	128.99	2.469	8393

Table 3: Execution time in milliseconds with different amount of instances

on-premise estimation. According to the model, it will give a decrease of 67% of emissions.

5.2 Execution Time

On a computer with 16 GB of ram and an Intel Core i5 processor, I tested the program with 4 and 212 instances as input. Each stage of the pipeline was separately timed and recorded. This can be found in Appendix A. For both of these, I ran the program 20 times. Of that, I calculated the min/max and average which can be found in Table 3. The 'Estimate Emissions' is the part CCFT normally does. On average for 4 instances the wrapper takes 98.0% of the time. For 212 instances it takes 98.6% of the total execution time. It shows clearly in the table that most of the execution time comes from the LookUp Table (LUT) creation. Just looking at the LUT creation we see an increase in execution time of 2722 %, while the amount of instances has an increase of 5300 %.

6 Discussion

Just like the results this is divided into two sections.

6.1 Output

The following aspects stand out when we compare the AWS instance with the original instance. Firstly, due to the filter, each important property is at least as good as the given configuration. Secondly, the case study required for the instances to be bare metal. AWS only provides bare metal instances which are the best in that instance family. Thus, resulting in an instance that is overpowered for the task at hand.

BT Global has a dashboard that provides real-time data about emissions from private cloud instances. The predicted emissions of the on-premise instance are in the same range as those reported by the dashboard. For that reason, these estimations, within a certain error range, can be assumed to be correct.

The reduction of emissions is according to the prediction 67%. Since AWS claims to have an improvement up to 93%, this number is possible. There might even be a better improvement due to the second aspect defined above.

6.2 Execution Time

Table 3 indicates that most of the execution time comes from the LUT creation. Even though the model takes more time with a higher amount of instances it scales up well. This means the configuration can have hundreds of instances, but a prediction of emission can still be given within a couple of seconds[4]. The model is thus usable in real-life applications.

7 Further Work

At the moment we do not know how reliable the model defined by the CCFT is. For that reason, multiple instances should be run on AWS and looked at the reported emissions by AWS. Based on that an error range can be given.

[4]Compilation not included

There was no time to also add support for Azure and Google Cloud services. This can be added to give more insight into which service gives the best emission results.

Moreover, the matching is done by a very crude combination of MAUT and AHP. More research could be done to determine the best way of determining the best instance mapping.

Lastly, the algorithm can be made quicker by preloading and calculating certain numbers. During the fitter the emissions are calculated for each instance left. At times these emissions are recalculated multiple times. Also, cache can be used to quickly get a response for often asked configurations. However, since the compilation time takes most of the time this should not be a priority.

8 Conclusion

This project looked at how to predict public cloud emissions based on private cloud configurations. A mix between AHP and MAUT was used for the purposes of mapping private cloud service configurations to their closest public cloud ones. Then the configuration was used to both get a local and public cloud emission prediction. With was done based on the values provided by the CCFT. This showed that in the specific use case an improvement of 67% would be made. This means based on this toolkit the greener choice would be to switch public cloud. We also showed that the execution is done within a reasonable time. Lastly, the argumentation about the emission values showed these numbers to be believable. However, until it is tested with a real AWS instance, there is no way to know for certain and know the error range.

A Execution Times

Run	On-Premise	Create LUT	Estimate emissions	Create report totals	Total execution time
1	7.668	210.125	5.64	0.424	262.872
2	7.116	195.264	5.504	0.517	247.283
3	6.949	193.026	5.734	0.543	250.646
4	14.684	1793	30.331	0.539	1913
5	7.492	198.651	4.758	0.428	250.581
6	8.397	198.944	4.695	0.577	251.677
7	7.444	197.249	5.207	0.699	250.12
8	7.305	194.005	5.578	0.508	252.893
9	7.86	204.571	5.084	0.482	256.244
10	10.085	227.199	5.484	0.524	294.871
11	7.04	198.247	5.081	0.5	252.501
12	7.662	192.978	5.224	0.503	244.713
13	7.129	209.248	4.98	0.53	263.28
14	6.503	189.357	4.743	0.405	238.9
15	7.879	207.651	4.638	0.505	263.651
16	7.583	225.489	6.051	0.475	286.878
17	8.483	249.339	12.785	0.847	320.396
18	7.598	205.085	5.367	0.51	259.017
19	7.425	202.281	5.151	0.517	260.46
20	7.638	195.007	5.201	0.437	247.011
Min	6.503	189.357	4.638	0.405	238.9
Avg	7.997	284.3358	6.8618	0.5235	343.3497
Max	14.684	1793	30.331	0.847	1913

Table 4: Execution Time of the pipeline for 4 Configurations

Run	On-Premise	Create LUT	Estimate emissions	Create report totals	Total execution time
1	36.365	7261	103.271	1.446	7523
2	25.971	7073	91.844	1.596	7272
3	23.13	7690	105.957	1.371	7905
4	24.282	7749	108.229	1.391	7969
5	27.465	7754	111.254	2.354	7976
6	25.6	7726	113.998	1.566	7954
7	23.457	7757	111.213	1.525	7980
8	24.772	7659	114.845	1.564	7888
9	23.83	7646	113.246	2.336	7876
10	25.264	7794	100.026	1.387	7999
11	28.971	8148	128.99	1.552	8393
12	23.83	7727	108.452	1.489	7940
13	24.79	7831	123.074	1.453	8072
14	24.383	7830	102.657	1.465	8045
15	24.333	7695	111.773	2.469	7930
16	24.697	7697	111.025	1.413	7914
17	24.664	7799	116.238	1.428	8023
18	29.313	8121	118.205	1.489	8365
19	36.657	7869	107.153	1.331	8127
20	24.605	7797	100.165	1.472	8011
Min	23.13	7073	91.844	1.331	7272
Avg	26.31895	7731.15	110.08075	1.60485	7958.1
Max	36.657	8148	128.99	2.469	8393

Table 5: Execution Time of the pipeline for 4 Configurations

References

[AMY10] Alireza Afshari, Majid Mojahed, and Rosnah Mohd Yusuff. "Simple additive weighting approach to personnel selection problem". In: *International journal of innovation, management and technology* 1 (5 2010), p. 511.

[Bel+08] Christian Belady et al. "Green grid data center power efficiency metric: PUE and DCIE". In: (May 2008).

[Fre+21] Charlotte Freitag et al. "The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations". In: *Patterns* 2 (9 Sept. 2021), p. 100340. ISSN: 2666-3899. DOI: 10.1016/J.PATTER.2021.100340.

[Gup+20] Udit Gupta et al. *Chasing Carbon: The Elusive Environmental Footprint of Computing*. 2020. DOI: 10.48550/ARXIV.2011.02839. URL: https://arxiv.org/abs/2011.02839.

[GVB11] Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. "SMICloud: A framework for comparing and ranking cloud services". In: *Proceedings - 2011 4th IEEE International Conference on Utility and Cloud Computing, UCC 2011* (2011), pp. 210–218. DOI: 10.1109/UCC.2011.36.

[IEA22] IEA. *Global CO2 emissions rebounded to their highest level in history in 2021 - News - IEA*. Mar. 2022. URL: https://www.iea.org/news/global-co2-emissions-rebounded-to-their-highest-level-in-history-in-2021.

[Inc22] Thoughtworks Inc. *Overview - Cloud Carbon Footprint*. [Online; accessed 20-June-2022]. 2022. URL: https://www.cloudcarbonfootprint.org/docs/overview.

[MHW08] H Scott Matthews, Chris T Hendrickson, and Christopher L Weber. "The Importance of Carbon Footprint Estimation Boundaries". In: *Environmental Science & Technology* 42 (16 2008). PMID: 18767634, pp. 5839–5842. DOI: 10.1021/es703112w. URL: https://doi.org/10.1021/es703112w.

[Myt20] David Mytton. "Hiding greenhouse gas emissions in the cloud". In: *Nature Climate Change* 10 (8 2020), p. 701. ISSN: 1758-6798. DOI: 10.1038/s41558-020-0837-6. URL: https://doi.org/10.1038/s41558-020-0837-6.

[Res19] 451 Research. "The Carbon Reduction Opportunity of Moving to Amazon Web Services". In: *Black & White paper* (2019).

[Roy68] Bernard Roy. "Classement et choix en présence de points de vue multiples". In: *Revue française d'informatique et de recherche opérationnelle* 2 (8 1968), pp. 57–75.

[Saa90] Thomas L. Saaty. "How to make a decision: The analytic hierarchy process". In: *European Journal of Operational Research* 48 (1 Sept. 1990), pp. 9–26. ISSN: 0377-2217. DOI: 10.1016/0377-2217(90)90057-I.

[Som+20] Emily Sommer et al. *Cloud Jewels: Estimating kWh in the Cloud*. Apr. 2020.

[Sun+14] Le Sun et al. "Cloud service selection: State-of-the-art and future research directions". In: *Journal of Network and Computer Applications* 45 (Oct. 2014), pp. 134–150. ISSN: 1084-8045. DOI: 10.1016/J.JNCA.2014.07.019.

[Wes21] Richard Westerhof. "Exploring the Factors of Sustainability and creating a Sustainability Dashboard for BT Global Services". In: *RuG Internship Report* (July 2021).

[WRI04] WBCSD WRI. "The greenhouse gas protocol: A corporate accounting and reporting standard". In: *World Resources Institute (WRI) and World Business Council for Sustainable Development (WBCSD), Washington, DC and Geneva* (2004).

[ZTK14] Edmundas Kazimieras Zavadskas, Zenonas Turskis, and Simona Kildiene. "State of art surveys of overviews on MCDM/MADM methods". In: *Vilnius Gediminas Technical University* 20 (1 2014), pp. 165–179. ISSN: 20294921. DOI: 10.3846/20294913.2014.892037. URL: https://www.tandfonline.com/doi/abs/10.3846/20294913.2014.892037.

[ZZZ09] Wenying Zeng, Yuelong Zhao, and Junwei Zeng. "Cloud service and service selection algorithm research". In: *2009 World Summit on Genetic and Evolutionary Computation, 2009 GEC Summit - Proceedings of the 1st ACM/SIGEVO Summit on Genetic and Evolutionary Computation, GEC'09* (2009), pp. 1045–1048. DOI: 10.1145/1543834.1544004.