Wind-Driven Clouds

Utilizing wind energy in data centers

Idun Osnes

Thesis submitted for the degree of
Master in Materials Science and Nanotechnology
60 credits

Department of Physics
Department of Technology Systems
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2018

Wind-Driven Clouds

Utilizing wind energy in data centers

Idun Osnes

Summary

This thesis is written as part of the project INTEGRARE (Intelligent prediction and integration of renewable energy sources into the Norwegian electricity grid), which addresses the need of transforming the future energy systems based on renewable energy sources. This project is a collaboration between several departments at UiO (the Department of Technology systems, the Department of Informatics, and the Department of Physics), as well as the Department of System Design Engineering at Keio University in Tokyo, and the Department of Energy Informatics at Technical University of Munich. This thesis aim to establish how power hungry data centers can be integrated within a smarter energy grid, considering that data centers have a good profile for using excess energy from renewable energy sources that would otherwise be wasted.

This thesis presents a method for reducing the environmental footprint of the data center industry, by creating an algorithm that reduces the overall energy consumption of a small-scale data center, co-located with a wind energy source. The algorithm also increases the fraction of energy that comes from the wind and decreases the energy consumption during times of low renewable availability.

The lower energy consumption of the data center is achieved by increasing the load on individual machines in the data center so that the overall efficiency of computations increase. The algorithm reduces the energy usage further in times of low renewable energy production, by separating the workload of the data center into different categories based on latency sensitivity, and postponing some or all of the lower latency sensitive computations to a time when more renewable energy is available.

Acknowledgements

First of all, I would like to express my gratitude to my supervisors, Associate Professor Sabrina Sartori at the Department of Technology Systems, and Professor Frank Eliassen at the Department of Informatics, and also to Prof. Dr. Arno Jacobsen from the Technical University of Munich. Thank you all for letting me follow my interests and take on this challenge, and for helping me through the process. Special thanks to my main supervisor Sabrina for her guidance and support to navigate the process of writing this thesis, it has been so valuable. Thank you for motivating me and pushing me further.

A big thanks to Associate Professor Anis Yazidi at OsloMet for being there with a smile to answer my questions and helping me set the course of my work, and for his valuable comments and input during this process.

I would also like to give a special thanks to Professor Toru Namerikawa and his students at the Department of System Design Engineering at Keio University for their hospitality, and for giving me the opportunity to visit and work in their lab in Tokyo.

I would also like to acknowledge how lucky I am to live in a country where education is free and available to all, and to mention my fellow students of the MENA-program, who have made the last five years of studying so pleasant, by creating a community for learning, discussions and good times. I am grateful to have gotten to know all of them.

Last but not least I want to thank my family. My parents for always supporting me and encouraging me to be curious and to figure things out on my own. My sister for her support, comfort, and understanding in the more challenging times, and a very special thanks to my girlfriend for her patience, and for everything that she has done for me during this process. Thank you so much for always having my back, rooting me on and giving me confidence. Also a shout out to my dogs for always being ready to cuddle.

Contents

1 Introduction **5**

1.1 Motivation 6

1.2 Problem Statement 6

1.3 Contributions 8

1.4 Organization 8

2 Background and state-of-the-art **10**

2.1 Background 10

2.1.1 Data centers 10

2.1.2 Workloads 12

2.2 State-of-the-art 13

2.2.1 Power sources 14

2.2.2 Approaches to process different workloads with RES 16

2.2.3 Approaches to optimize scheduling software 18

2.2.4 Approaches for power and workload prediction 21

2.3 Methods 25

2.3.1 Neural networks 25

3 Design of experiments **30**

3.1 Design choices 30

3.1.1 Workload trace 30

3.1.2 Assumptions and simplifications 32

3.1.3 SVM regression for wind power prediction 34

3.2 Data description 34

3.2.1 Google cluster-usage traces 34

3.2.2 Data selection and preprocessing 35

3.2.3 Workload statistics 36

3.2.4 Wind data from a Norwegian wind farm 38

3.3 Description of the proposed algorithm 39

3.3.1 Baseline algorithm, assuming clairvoyance 40
3.3.2 Baseline algorithm with power, assuming clairvoyance . . . 41
3.3.3 Non clairvoyant algorithm with task usage prediction 44
3.4 Neural network for task usage prediction 45
3.4.1 Data preparation and inputs 45
3.4.2 Network structure 46

4 Results and discussion 49
4.1 Original data 49
4.2 Results from baseline algorithm 52
4.2.1 Unaltered trace 52
4.2.2 Custom trace 55
4.3 Results from power-aware algorithm 57
4.3.1 Unaltered trace 57
4.3.2 Custom trace 59
4.4 Results from algorithm with predicted CPU usage and duration . . 61
4.5 Comparing the results 65
4.6 Task prediction results 67
4.6.1 Prediction for low latency sensitive tasks 67
4.6.2 Prediction for medium latency sensitive tasks 69
4.6.3 Prediction for high latency sensitive tasks 70

5 Conclusions and further work 73
5.1 Conclusion 73
5.2 Further work 75

Chapter 1

Introduction

We live in a world that is more and more dependent on information technology (IT). The global interconnectedness and mass accumulation of data is a growing business, and growing fast. There are few areas of our life in today's world that are not touched in some way by technology. This technology is getting smarter and more interconnected each day, by collecting, processing and analyzing vast amounts of data. Computers control much of our infrastructure. The energy-grid, water supply, financial institutions and most other critical systems now run on some form of a connected system. These systems are dependent on a growing number of computers and servers, located around the world. A majority of the population of the world also use the internet, connecting to the world with their phones, tablets, and laptops. As a result of this, there is an increase in demand for reliable, secure data centers, to process and store all this data.

Data centers use a lot of power for processing information. The processing generates a lot of heat, which means the data centers also use almost the same amount of power for cooling. In 2010, it was estimated that data centers used 1.5% of the worlds total electricity consumption, and this is a continuously growing number [1].

All this contributes to the huge energy consumption of the IT-industry. In 2016, a report from Greenpeace estimated that the combined IT-sector stood for about 7% of the worlds total electricity demand in 2012, and projected that this could grow by 7% each year through 2030 [2]. It is not an easy task to estimate exactly what fraction of this energy is consumed by the data centers themselves [1]. This is due to a set of different factors, one being the rapid advancement in technology and software, another being the secrecy many of these companies have around their business.

A thorough study of previous research done on communications technology electricity consumption estimates that in 2012, data centers alone consumed 281 TWh [3]. According to the International Energy Agency, the worlds total electricity consumption in 2012 was 20919 TWh, which gives the data centers a total of 1,34% of the worlds total electricity consumption in 2012 [4]. This is quite significant, and there is no sign that this number will decrease over the next decades. The same study projected that in 2016, data centers would consume 474 TWh, while according to Enerdata [5], the world consumed 21191 TWh, which gives 2,2% of the world total electricity consumption.

1.1 Motivation

To handle all this demand, there is a need for sustainable, smart solutions on both how to get the power needed, and also to efficiently handle the increasing workload these data centers manage every day. Renewable energy sources (RES) are making their way into the energy market at a rapid pace. While this is good news with regards to CO_2-emissions and sustainability, there are challenges with integrating these highly fluctuating power sources into the energy grid. Peak production of renewable power, does not necessarily match up with peak electricity demand, and even if it does, traditional power plants running on non-renewable fuel, cannot be shut down and powered up quickly or cheaply. As a consequence of this, some of the produced renewable power ends up going to waste.

This challenge poses an opportunity for data centers. If the stranded power could be utilized in the power-hungry data centers, this would benefit both the power utility companies, and the data centers. In addition to this, it could help stabilize the power grid, and will help reduce CO_2-emissions [6].

1.2 Problem Statement

This thesis attempts to identify approaches that would allow cloud data centers with diverse workloads to leverage access to wind energy, and reduce their power costs.

Globally, the power system today is largely based on brown energy, that is energy produced from oil, coal, and gas (and to some degree nuclear energy). This energy production is stable, adjustable and predictable, which is essential in current data center operations, where continuous operation is demanded, and where a power outage could be disastrous.

On the other hand, renewable energy resources are much more fluctuating and less predictable. Solar energy is only available when the sun shines, and wind only when the air is moving. To be able to utilize this unstable and unpredictable energy in data centers, either a huge amount of energy storage needs to be accessible, or the methods of how workloads are handled in the data center need to be addressed. The latter is what this thesis is focused on.
If we could handle fluctuating energy, by predicting both available power and estimated workload, and schedule this accordingly, there is a chance that the data center industry could grow with a more sustainable profile. Previous research on workload optimization in data centers with renewable energy have mainly focused on one of the following aspects:

- Utilizing excess energy by starting up servers when energy is available, by having clusters of servers that handle non time-sensitive workloads, that can be paused [6].

- Processing batch workloads in environments with solar power, combined with batteries and net-metering, to achieve a net-zero consumption of brown energy [7].

- Using green energy by having geographically separated data centers, using virtualization to migrate workloads between them, based on available energy at each site [8]

Previous research has mainly identified approaches to utilize green, unpredictable energy in data centers that handle a more stable demand. In this thesis, a combined workload representing a variable demand is analyzed and tested for optimization with use of wind energy. The goal of the thesis is to identify methods for data centers handling a variety of workloads to utilize unpredictable wind energy sources. We aim to reduce the overall electricity consumption of the data center and try to shape the workload in the data center to follow the production of available renewable energy.

1.3 Contributions

This thesis proposes to create a workload scheduling system for a data center, co-located with a wind-energy source. The system takes into account available (predicted) wind-energy and predicted workload and the goal is to optimize the allocation of tasks on the machines in the data center, to reduce brown energy consumption, by reducing the number of servers running, and being aware of renewable energy availability. This while still making sure that the demands from the customers are met. To achieve this, the model combines methods from several other scheduling systems found in the literature. For example, it takes inspiration from a queuing system that splits the task into separate queues before placement [6].

This work proposes a workload scheduler applied to a simple, small computer cluster. The computer cluster handles a combined workload with a goal of utilizing as much wind energy as achievable and reducing the load to as few machines as possible. The scheduler performs this by separating the workload tasks into different priorities and placing them on a number of machines based on available energy. First, it estimates the energy needs of the incoming tasks and determines the latency sensitivity of the individual tasks. Then, estimations on duration and CPU resource usage for the individual tasks are predicted with a neural network. Based on the current workload on the machines, and the estimated available energy, the scheduler places the tasks in different queues. The higher priority tasks are put on some machines, while the ”free” priorities are scheduled according to time sensitivity and energy availability. Some machines are shut down when there is a lack of energy available, but there are always machines running, to keep the high-latency sensitive tasks running at all times. Evaluation results show that the proposed scheduler can reduce the number of running machines at times of low wind energy availability up to 79% of the time. We also see a reduction in total energy consumption and a slight increase in the fraction of total energy consumption that comes from the wind.

1.4 Organization

The thesis is structured as follows: Chapter 2 presents background information about data centers, workload and energy profiles, related literature and introduces the methods used for workload prediction. Chapter 3 discusses the experimental work, and gives detailed descriptions of the design choices and implementation of the developed algorithms. In chapter 4 the results are presented and discussed. Finally, chapter 5 summarizes the conclusions from this work and presents an outlook for further work.

Chapter 2

Background and state-of-the-art

This chapter provides background information about data centers and workloads, beneficial to understand the problem and solution proposed in the following chapters. State-of-the-art research on the subject of utilizing different renewable energy sources in data centers is also presented. At last, some of the central methods used in this work is described.

2.1 Background

2.1.1 Data centers

“Data center” is a term that can apply to many different types of computing clusters. In general, the term is used to describe any room or housing of servers and communication systems. Most companies have their own data centers, used to store and process their data and communications. With the growth of web-services, there is also a growth in data centers that rent out servers and data rooms, so that businesses no longer need to maintain all these servers themselves.

Cloud data centers

With the enormous growth of the internet and web services in the last decade, many companies have specialized in data handling. Google, Amazon, Facebook, and Microsoft are some examples of such companies. These companies have huge data centers, consisting of tens of thousands of machines, that handles and processes a lot of different types of data or workloads, such as web-search, email services, financial computations, streaming of video and so on.

The data centers handling all this data are necessarily huge and complex. They also consume a vast amount of electricity. Not only the processing itself consumes energy, the infrastructure to run such a huge amount of servers is also energy costly [9].

Power usage of data centers

The enormous energy consumption of data centers is related to their servers running at all times. However, this is not the only thing in a data center that consumes power. Storage and memory usage, networking and cooling also use a whole lot of energy [9]. The energy efficiency of a data center is hard to measure, because of the difference in workload and applications in different data centers, and also because of the different hardware [9]. One common way to measure the energy efficiency is by calculating how much of the total power is consumed by the actual computations, the Power Usage Effectiveness (PUE) 2.1.

$$PUE = \frac{\text{Facility power}}{\text{IT equipment power}} \tag{2.1}$$

This measurement is widely used in the business, to compare the efficiency of different data centers. A survey from the Uptime Institute found that the average PUE of the participating data centers in 2017 was 1.7 [10]. It is worth to note that the PUE is not an optimal assessment of the actual energy efficiency since many results are based on optimal running conditions and best performance values [9]. The PUE has decreased in the previous years because of more efficient hardware, and also because there has been a lot of advancement in cooling technology in the data centers.

One of the issues with processing is that the server uses a lot of power, even when running at low utilization [11]. Many of the hardware upgrades are focused on reducing the idle power draw of the servers. Another way to improve on the issue would be to utilize more of the server capacity, thereby running fewer servers with higher utilization to increase efficiency. Combined with a secondary low-power state with a low idle power draw, the efficiency could improve even further.

The software that runs the data centers have not kept up with the hardware upgrades, and there is a lot of potential to increase efficiency using better strategies for workload planning and scheduling.

2.1.2 Workloads

In this thesis, the workload profiles of the data centers are a central part. There is no convention in classifying workloads in the literature since they are so diverse, but as a rough division, we define three groups: High-Performance Computing (HPC), batch, and interactive workloads. What separates the three, is the frequency of incoming requests, duration/computational demand and the latency sensitivity of the tasks. Traditionally, there has been a separation between these different workloads, because servers have been designed to handle the different types.

Recently, it is getting more and more common that businesses rent servers with companies that provide the hardware and infrastructure, in so-called cloud data centers. This opens up for a higher average utilization of each server, where batch jobs can run when the computational resource demand for the other services are low, such as in the middle of the night, or on holidays and weekends.

High Performance Computing (HPC) Workloads

HPC workloads demand a lot of processing power and usually have a good profile for parallelization, meaning that they can be spread over many cores, and can be paused and started again without losing progress. These tasks often solve computationally and data-intensive problems, like simulations, scientific calculations and so on. They allow a long response time, which, along with the parallelization, opens the possibility to postpone some or all of the calculations, and therefore have a good profile for using power from intermittent renewable resources.

Batch workloads

The batch workload is less computationally demanding than the HPC, but still needs a lot of processing power, and is usually done by parallel processing of different tasks within the jobs. The batch load jobs have a run-time from tens of minutes to several hours, and often have a deadline of 24 hours or less after delivery. These jobs can, for instance, be calculations on the number of sales from a retailer, updating account balances from a bank and so on.

Interactive workloads

The interactive workloads are the loads that require the lowest response time. Typically these are requests from web services, that have a varying amount of requests through the day, and each task requires a response time in the order of less than a second to tens of seconds. The number of tasks from these workloads follow a repetitive pattern during the day and week (see figure 2.1), with a peak mid-day, valley in the early hours and lower overall demand in the weekends [12], but can be unpredictable in smaller time periods.

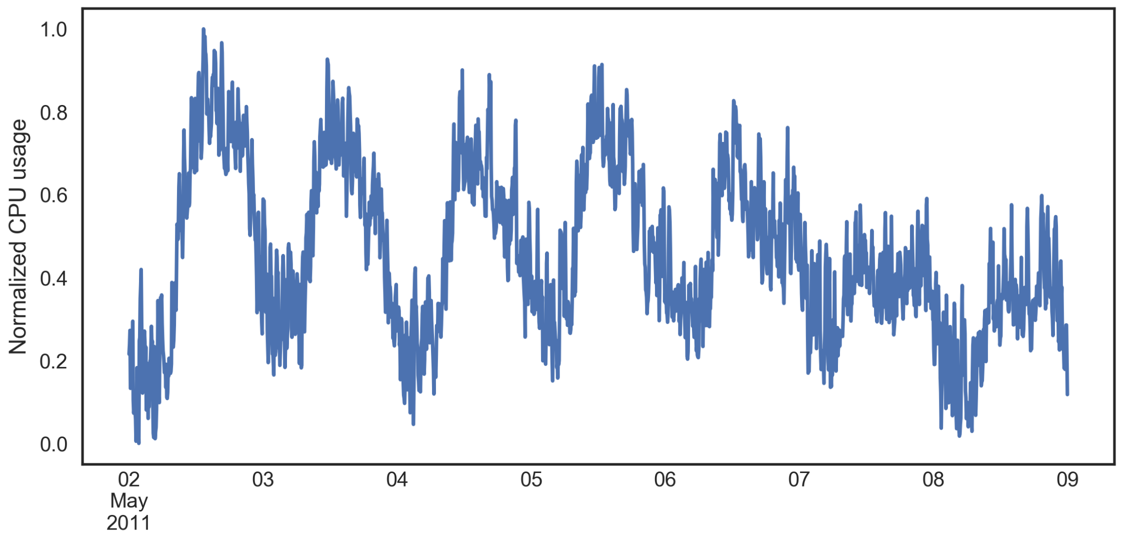

Figure 2.1: Interactive workload profile over a week.

2.2 State-of-the-art

In 2010, the energy consumption of data centers was estimated to be about 1.5% of the worlds total energy consumption. Even though there have been advances in the efficiency of the data centers, this number has increased and will most likely continue to do so [1]. Figure 2.2 illustrates the growth in network traffic from 2011 to 2016. About 70% of total traffic is within or between data centers.

This growth, along with the growing awareness of how fossil fuels impact our environment, makes for good incentives to exploit renewable energy in data centers. The topic has been the subject of several publications.

In this section, different approaches from the literature are presented, on how to utilize power from renewable resources to process different types of workloads.

The presented literature was chosen to show different approaches to solve the problem of using renewable energy sources in data centers.

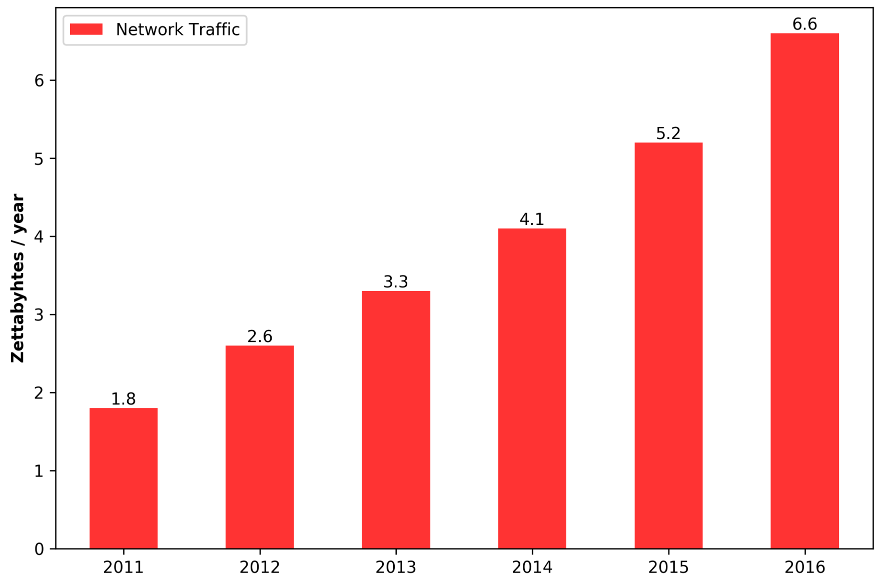

Figure 2.2: Growth in core network traffic, including traffic within data centers. Adapted from [1].

2.2.1 Power sources

Traditionally, data centers are connected to the grid, which is mainly (at least in many areas) driven by fossil fuels. The power grid is stable and predictable, and so are the prices that follow the demand. Figure 2.3 illustrates an example of the fluctuating power output from wind and solar generation over a week. Solar power has a more predictable daily pattern of power generation, while wind power does not show any obvious pattern. If the excess power that is produced at some times could be utilized, for example by co-locating the power source with a power demanding data center, both the power source and the data center could benefit. To utilize an unreliable power supply, there is a need to adapt the power consumption to these intermittent sources. One way to do this is by creating so-called energy agile workloads, that adapt to a variable supply of energy [7].

Aside from the aforementioned challenges, there are several advantages to using renewable power. Even though the initial cost is high, the price of the renewable energy after installation is low and predictable [13]. Another advantage is the modularity of renewable sources: It is simple to add more power generation if the demand for power increases. This, in combination with the data centers option to increase servers and computing power, makes for a flexible system.

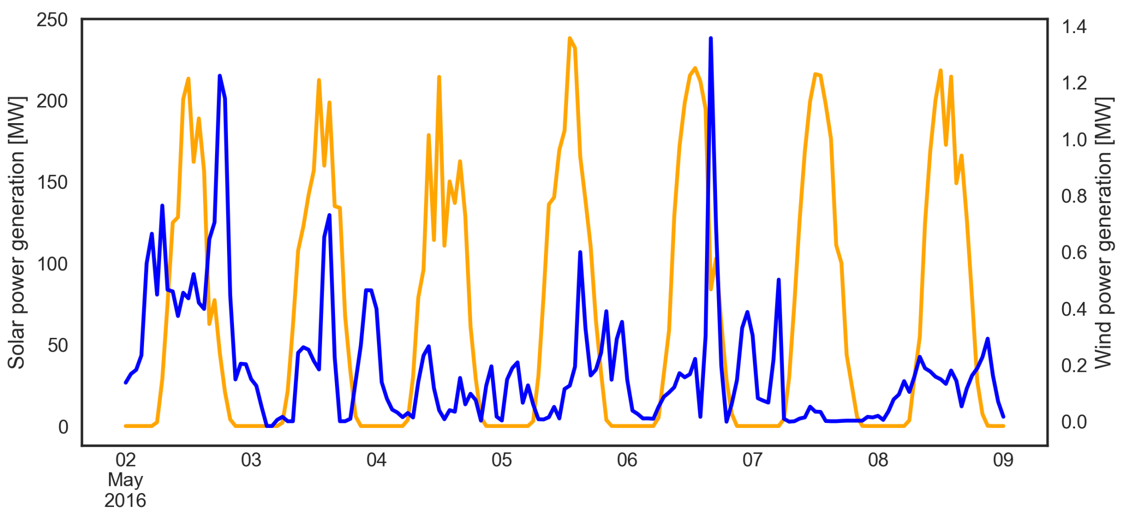

Figure 2.3: Renewable generation over a week, wind data from a single wind turbine in Norway, solar data from Belgium [14].

Solar power and data centers

The majority of research on renewable energy sources combined with data centers is done on systems using solar power as the renewable source. There are several reasons why this is the case. One is that solar power is an established renewable resource, projectively reducing its cost [15]. The generation of solar power is also more or less predictable, compared to other renewables like wind power (illustrated in figure 2.3). Another important point is that the solar production pattern matches the typical interactive workload demand pattern as can be seen by comparing figure 2.1, which illustrates the demand for processing power for an interactive service, and figure 2.3 which illustrates renewable power generation. With this said, there is also an obvious challenge with solar power. Data centers use a lot of energy for cooling, as the servers produce a lot of heat when processing [9]. Cooling is cheaper when it is cold outside, typically at night when there is no solar energy production.

Íñigo Goiri et al. [7], proposed a system applied to an actual small-scale data center powered by solar panels, with the option to store excess power in batteries, or sell it back to the grid by doing net metering[1]. The goal was to achieve a net-zero exploitation of non-renewable power sources. The use of batteries for storage of excess power might not be the most efficient, though. A research comparing the

[1]Net metering is a system where excess renewable power is transferred into the grid, and the consumer is compensated. In this way, the consumer only pays for the net energy consumption from the grid.

use of battery storage and the use of energy agile loads estimated that 200 tons of lead-acid batteries would be needed to achieve the same renewable utilization [16].

Wind power and data centers

Wind power is more unpredictable than solar power. There is no clear pattern in its fluctuation, as there is with the daily variation of solar power, as can be seen in figure 2.3. This poses a challenge for both predicting the supply and also for adapt the workloads to a fluctuating power supply. One possible solution is to use geographically different wind parks co-located with data centers and connect them using virtual machines [8]. Another way is to use excess energy, like stranded power used to compute HPC workloads [6], or to have two different computing clusters in the data center, connected to different power sources [17].

Other renewable energy sources

To my knowledge there is a lack of literature discussing the exploitation of other fluctuating renewable power sources than solar and wind in combination with data centers.

There are other ways to get the energy from renewable sources, for instance by purchasing clean power certificates from power companies, doing net-metering or by using batteries to store excess energy. Among other goals, one optimization goal is to achieve net-zero consumption. This is done by doing net-metering when the excess power is not consumed by the data center at the time of production in combination with batteries on site. A combination of these two gives good results on lowering the non-renewable energy consumption [7].

2.2.2 Approaches to process different workloads with RES

There are a wide variety of different computing tasks that are handled by data centers. Most data centers process many of these tasks in parallel and therefore need to be able to handle different workloads simultaneously. In the literature there are many different classifications of workloads, here they will be loosely categorized, as the definition of the different loads overlaps in many ways. The three categories of workloads that are presented are the same as we introduced in section 2.1.2; high-performance computing, batch and interactive.

High-Performance Computing (HPC) workloads

Different optimization goals are attempted in different approaches. The studies presented here tries to leverage access to renewable energy, but the cost function differs. It could be the price of electricity (often cheaper when renewable energy is available) [7], or there could be value in using as much renewable as possible [6], or to reduce the use of brown energy [17]. One goal is to minimize the use of brown energy. To achieve this, HPC jobs could be handled by increasing parallelization, thereby speeding up the tasks by using more processors to compute the job, as in the work of Md E. Haque et al [18]. When excess renewable power is available, the HPC workload is spread over more cores, so that the job can complete faster. This builds some slack for the task to finish on fewer processors if the renewable supply should drop.

Fan Yang and Andrew A. Chien presented a method using stranded power[2] to process HPC tasks [6]. Here, the servers that compute the tasks, are only turned on when stranded power is available. The clusters of servers that handle the HPC tasks are connected to wind turbines. They are also connected to a central data center, which in turn is connected to and powered by the normal electricity grid. The study showed good results for exploiting the excess energy (with an availability of stranded wind power up to 80% of the time).

Batch workloads

In many studies, a combination of batch and interactive workloads are tested [7, 12, 19, 16]. The batch workloads often have deadlines in order of hours, so they are possible to defer to times when energy is expected to be cheaper or more renewable energy is available [7, 19], or to shift the computations to night-time, when the required power for cooling is reduced [12].

[2]Stranded power is excess produced energy that for some reason cannot be put into the energy grid, and otherwise would be wasted.

Interactive workloads

Methods for efficiently handling interactive workloads are proposed only in a few papers [7, 16]. An interactive workload (explained in section 2.1.2) could for example be a trace from Facebook [7], or a trace from Wikipedia [16]. Different methods are used to handle the interactive workload, a scheduling of battery and net-metering is used to minimize grid power (the interactive requests are handled as normal) [7], or a degradation of performance of the interactive workload, by applying some of the same strategies that are used when handling flash crowds (an unexpected spike in site traffic) [16]. The quality degradation includes using cached sites, not fully loading page elements and so on. In other works, the interactive workloads with low response time are prioritized, and no deferring or quality decrease is performed on them [6, 12, 19].

2.2.3 Approaches to optimize scheduling software

There is obviously a huge demand for good methods to reduce the energy consumption in data centers. When over 1.5% of worldwide electricity is used for computing [1], a value expected to increase, it is essential to lower the electricity consumption, both for cost reduction and for lowering the environmental footprint of the industry. A lot of research is done with the aim of improving the hardware in the servers, and the cooling infrastructure in the centers. Instead, in this thesis, the focus is on how to optimize the software that handles the workload, so that it can better exploit green energy. There are different approaches possible, some are discussed below.

1) Workload migration

Workload migration within a single data center consists of having several computer clusters, some connected to the grid, others are connected to a renewable power source, for instance, a wind-power source, as illustrated in figure 2.4 [17]. The system uses virtual machines to migrate the workload between the server clusters. All the transitions are recorded and taken into account when producing a utilization schedule so that the servers capacity to do work is not disturbed by rapid transitions.

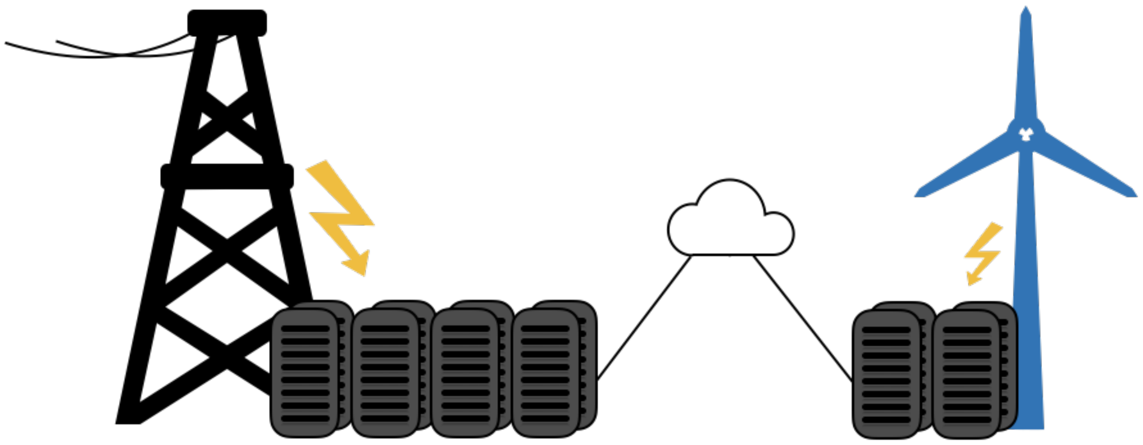
Figure 2.4: The majority of the servers could be connected to the energy grid, while smaller clusters are powered by wind turbines [6].

2) Performance degradation

To lower the energy cost of a task, one method is to opt for lower quality of the result [16]. There are different types of quality decrease for different types of workload. For batch workloads, an example of quality reduction is postponing the execution of tasks. Figure 2.5 shows how the processing of the load is scheduled at a time with either low cost of brown energy or when there is an excess of power from the renewable resources.

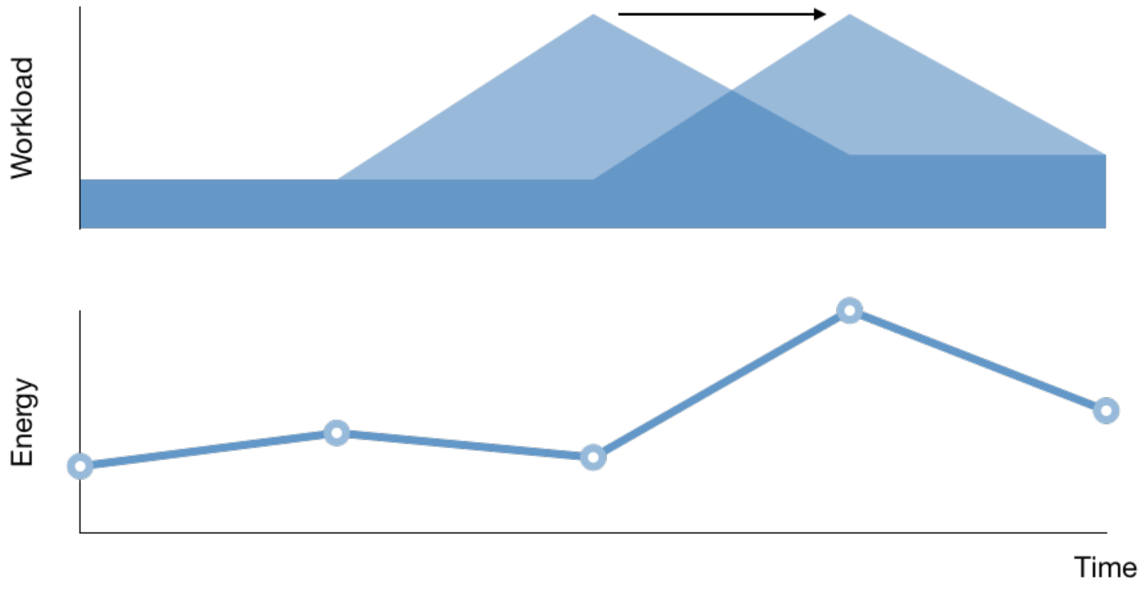

Figure 2.5: In batch workloads, the peak demand can be shifted to match the peak energy production.

In the interactive workloads, the quality reduction will differ depending on the type of trace. For web-services, there is already a way to reduce the quality, by using the same methods as when the servers handle flash crowds.[3] Strategies for handling flash crowds could be to use cached sites or partially load page elements (for example reducing the quality of images) [20]. The same methods could be used to produce a reduced quality of response in the event of a scarce renewable power supply. The cluster manager has as a goal to maintain a stable energy cost. It does this by calculating what fraction of the incoming loads that should be marked as degraded, and then the cluster manager marks that fraction of incoming request with a marker to be degraded. The fraction, f_d is determined from how much energy would be saved by degrading the load:

$$f_d = (1 - \hat{L})/(1 - E_d) \tag{2.2}$$

where $\hat{L}$ is the desired load as a fraction of the old load, and E_d is the work required to process a degraded request (as a fraction of normal request).

3) Other

One way to exploit the option to defer the batch tasks, while keeping the interactive tasks unaffected [19] is to put the batch and interactive tasks are in different queues. In this way, the interactive workload is not disturbed by the batch computing, thus managing to keep the strict response-time of the interactive load.

Another approach that is worth mentioning, is geographically distributed data centers. The idea is that several different data centers can migrate workloads using virtual machines, quite similar to the approach mentioned above. With each data center (or server cluster) co-located with different renewable resources, the supply of power could be more or less stable and predictable when combined. For instance, two solar-powered data centers in different time-zones could cover more of the day with a high output of power, or as in figure 2.6 below, the data centers could be connected to different renewable energy sources. The challenges then lie in optimizing the migration strategies [8].

[3] A flash crowd occurs when a lot of people try to access the same server at the same time. An example could be when a small website is linked to from a website with a much larger audience, causing an unexpected spike in traffic to the smaller site.

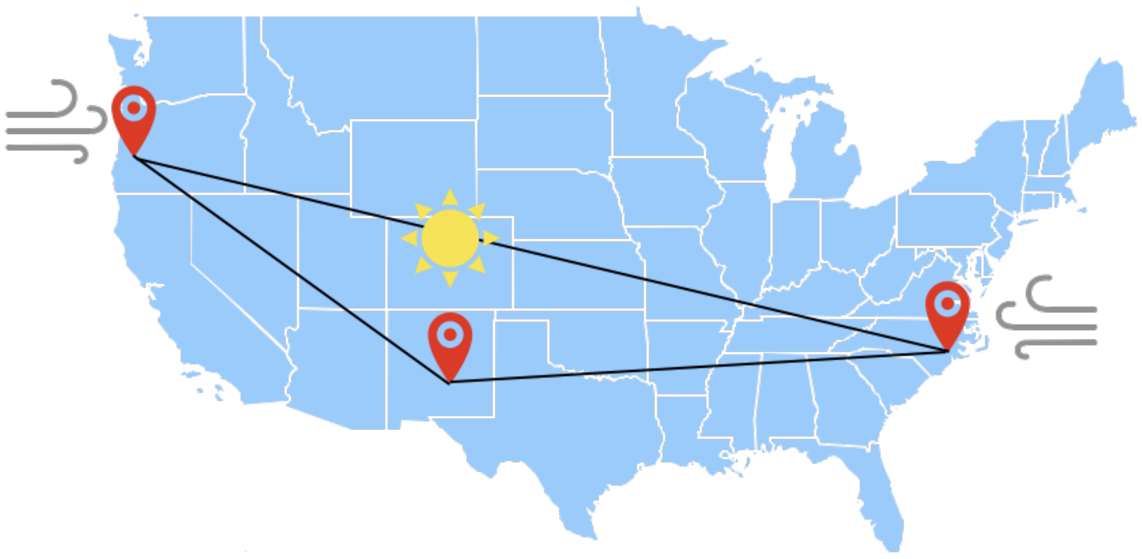
Figure 2.6: Workload migration in geographically distributed data centers.

2.2.4 Approaches for power and workload prediction

There are a lot of different approaches to power and workload prediction. Some studies attempt to schedule and predict batch and interactive workloads, some only schedule the batch load, while others refrain from predictions or schedules at all, and only have a reactive response to the renewable power supply. The methods used in the assessed papers are described below. These articles were chosen because they represent different approaches to solving the problem of utilizing renewable energy in data centers. They have a varying complexity of prediction methods both for power and workloads.

Power prediction methods

The prediction of renewable power production is a difficult task, and it is out of the scope of this thesis. The methods used for power prediction in the discussed articles will be presented regardless, for completion.

In the work of Zhenhua Liu et. al. [12], the power prediction used is similar to Support Vector Machines (SVM) with an RBF kernel. A k nearest neighbor based algorithm is used, with a 1-hour granularity of the forecast. It works by finding the most similar days from recent past, and then use the generation from these days to estimate the power output for the desired prediction interval. The prediction is quite good, with an error of 5-20%.

Simple methods for power prediction are also effective [7]. As long as the scheduling is performed regularly with short intervals, there is no need for complicated methods. Íñigo Goiri et al calculated the solar output with a simple equation (2.3). Here, $B(t)$ is the amount of energy generated on the day with the highest generation the previous month, and $CloudCover$ is the forecasted percentage of cloud cover (this is gathered from the weather forecast).

$$E_p(t) = B(t)(1 - CloudCover) \tag{2.3}$$

The system also makes a prediction for $CloudCover$ based on the power generation in the previous epoch, and then chooses the most accurate one for the next prediction interval.

A weather-conditioned moving average (WCMA) can also be used to predict solar power production [19]. As discussed above, simpler models with lower complexity and shorter time horizons are preferred also here. The WCMA algorithm performed well, with a mean error of 9.6% for the predictions [19].
For the wind power, wind speed and direction are used to predict the power output. Weighted nearest neighbor tables are used to generate wind power curves. The table is updated using the current information:

$$P_{new}(v, d) = \alpha * P_{obs}(v, d, t) + (1 - \alpha) * P_{old}(v, d) \tag{2.4}$$

where P_{new} is the new power curve table entry for given wind speed v and direction d. P_{old} is the existing value for the same direction and velocity, and P_{obs} is the observed value at time t. The authors of the study have used $\alpha = 0.75$, to favor the currently observed data. The prediction of the future interval is then calculated through the following:

$$P_{pred}(v, d, t + k) = P\big(v(t + k), d(t + k)\big) \tag{2.5}$$

This simple model performs well for the short intervals considered (mean error 17.2%) [19].

Another method is to closely monitor the price/availability of power and then adapt the received workloads to the available energy, by using the slack mentioned in section 2.2.3 [16]. A fraction of the incoming requests are marked as degraded, and these tasks get a simplified response. Instead of predicting the wind power, the characteristic Rayleigh distribution equation (2.7) for wind power is divided into three regions [17], as shown in figure 2.7.

$$f(v) = \left(\frac{2}{c}\right)\left(\frac{v}{c}\right)e^{-(\frac{v}{c})^2}, v \in [0, \infty) \tag{2.6}$$

In equation (2.6), c is the scale parameter, v is wind speed and f is frequency.

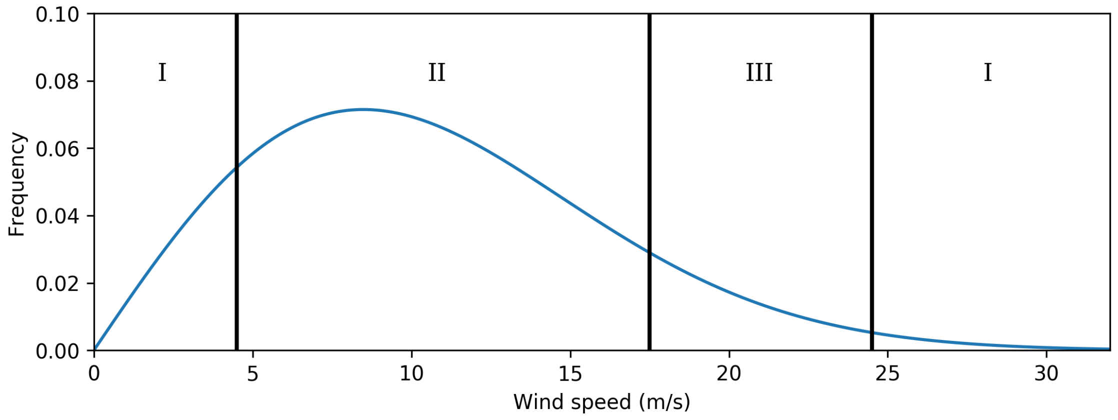

Figure 2.7: Rayleigh distribution of wind split into three regions. Adapted from [17].

Region I have no power output, while region II and III has a fluctuating output, so when the power supply is in this region, the power is tracked by lazy tracking (only the relatively stable power output is monitored). Every 15 minutes, a schedule for what servers should be powered by renewable sources are calculated and put to use. A renewable utilization of 94% is achieved with this configuration [17].

In the work of Fan Yang and Andrew A. Chien [6], there is only a reactive response to the power output and no real power prediction. The computing cluster connected to the wind park is fully switched on and completing tasks when there is stranded power available, and fully off otherwise.

Workload prediction methods

There are many different ways to perform workload prediction. As in the prediction of power, some don't predict the workload at all, but only react in a given way depending on what power is available at the arrival time [6, 17], while others have detailed prediction models for both batch and interactive workloads [7, 16].

One method for predicting workloads is pattern matching. To predict the resource demand for interactive computation, an analysis of the historical data can be used to find long and short-term patterns [12]. Fast Fourier Transform (FFT) is used to find the periodogram. A daily pattern is found from the periodogram, and the pattern is captured by using an auto-regressive model. The demand w at the time

t in day d is calculated from previous demand from N days, as seen in equation (2.7) below.

$$w(d,t) = \sum_{i=1}^{N} a_i * w(d-i,t) + c \tag{2.7}$$

Here, a_i is the IT capacity for an interactive workload i. The parameters are calibrated using historical data. The batch workload is predicted using historical data to find a ballpark approximation.

As mentioned, not all proposed methods predict workloads. An example is in the ZCCloud system [6], where the system is only fully on or off, using stranded power. Another method is to not predict the workloads, but separate the different jobs at arrival [19]. Here, the batch jobs are assigned to some servers, and have one queue, while the interactive requests are immediately handled, and distributed to the server with the least amount of batch processes running, to make sure everything functions optimally.

Systems can use workload prediction to determine how much power is needed in the coming time, to prevent the system from having latency if the rate of requests should increase [16]. An accurate prediction can bring up systems before needed, and take them down when not. Amongst other prediction methods, last arrival, moving window average and exponentially weighted average have been tested [16]. The exponentially weighted average with α = 0.95 performed best, with less than a 4% mean error.

The GreenSwitch system also uses an exponentially weighted moving average to calculate average power consumed in the past and uses this to predict the next interval [7]. It could be beneficial to use more sophisticated methods for the prediction, but this gives another challenge with the computation of complex models that takes both time and computational resources.

In the iSwitch system [17], the average load utilization is computed and recorded in each interval, and then used to predict the load in the next period.

The GreenPar system is tested with different information about future events (jobs and their speed up profile, energy availability and so on) [18]. There is no real workload predictor described in the paper. In only one of the four optimization strategies is there an implementation to include new jobs. The method is used just to re-solve the optimization periodically, to include new arrived jobs (24-hour scheduling horizon).

To my knowledge, none of the related works presented in here did any prediction on individual task usage. In the work of this thesis, a neural network is used to predict CPU-utilization and duration of individual tasks, instead of predicting the total predicted workload. A neural network was chosen due to the ability of a neural network to solve regression problems [21]. The main goal of the thesis is not to determine the best prediction method for CPU usage and duration of a task, but rather to identify how a non-clairvoyant scheduler best could place tasks on machines to reduce non-renewable energy consumption.

2.3 Methods

2.3.1 Neural networks

A neural network is a machine learning method used to predict a desired outcome based on an input vector with features. It learns by training on labeled data, which means that it is a supervised learning method. Neural networks can be used to predict a class if classification is the target, or it can be used for solving regression problems. According to the universal approximation theorem, a feed-forward neural network with one hidden layer having a finite number of nodes can approximate any continuous function [21].

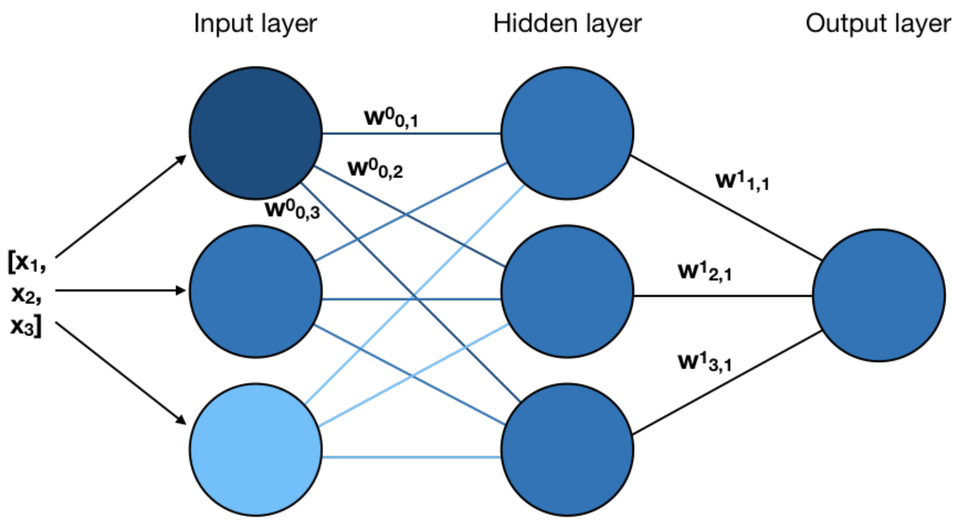

Figure 2.8: An illustration of a simple neural network with one hidden layer. x is the input to the network, w^n is the weight of the connections from layer n to layer n+1.

In this work, a neural network is used to approximate the function determining the mean CPU usage of a task, given input parameters gathered from the request of the task.

The network consists of layers of nodes, all connected with weights, w, as shown in figure 2.8. The first layer is the input layer, where the features of the training data are inserted. The last layer is the output layer, which gives the predicted value.

The workings of a neural network

The network learns to approximate the function, by adjusting the weights in the connections between layers. This is done using backward propagation. The weights are initialized from a normal distribution with mean 0 and standard deviation 0.05.

The prediction of a value happens by propagating the features forward through the network. First, the features in the form of a vector x, is given to the input layer of the neural network. Then, the features are multiplied by the weights connecting the nodes to the next layer and summed, according to equation (2.8),

$$z_j^L = \sum_{i=1}^{n} w_{i,j}^{L-1} * x_i \tag{2.8}$$

where, n is the total number of nodes in the previous layer, L is the layer, x_i is the input from node i in the previous layer, and j is the receiving node.

After the inputs are summed, an activation function determines the value of the node. The activation function can be expressed by a range of equations. In the network implemented in this thesis, the rectified linear unit function (ReLU) is used in all layers, as in equation (2.9).

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.9}$$

The value node j in layer L is then given by equation (2.10)

$$a_j^L = f(z_j^L) \tag{2.10}$$

where z is calculated by equation (2.8).

As the values propagate forward through the network, the activation function determines the relative importance of each node.

The value from each node is then sent forward through the network, and multiplied by the weights in the next layer, following 2.8.

This continues until we reach the output layer. The activation function in the output layer is a normal ReLU. This is to avoid negative values.

After the samples have been propagated forwards through the network as described above, the error of the prediction is calculated by a loss function. As with the activation function, the loss function can be represented by a variety of different functions, depending on the nature of the problem. For regression problems, the mean squared error and the mean absolute error are common loss functions. Equation (2.11) presents the mean absolute error function.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_{pred,i} - y_i| \tag{2.11}$$

Here, n is the number of samples, $y_{pred,i}$ is the predicted value, and y_i is the true value for sample i. This is the function used in this work.

To learn, the neural network needs to update the weights between the layers, to reflect their individual contribution to the error. This is done with a method called backpropagation, where the gradient of the loss function with respect to each weight, is used to update the weight.

Adaptive moment estimation optimizer (Adam)

How the weight updates are calculated depends on the optimizer. The goal is to reduce the loss function as much as possible, or in other words, to get the prediction error as low as it can be. The Adam optimizer has been shown to be a good optimizer for a variety of different neural networks and problems [22].

The weights are updated by taking the gradient of the loss function, in respect to the weights. This gives an estimate of how much each weight contributes to the overall prediction error.

The weights are updated by first calculating the gradient of the loss function, $E = MAE$, with respect to each weight, as in equation (2.12):

$$g_{t,i} = \frac{\delta E}{\delta w_{t,i}} \tag{2.12}$$

where t is the current ”timestep” or iteration.

To find the contribution of error from the different layers and nodes (and thereby the weights), an error function is needed. In the output layer, the error is given by equation (2.13).

$$\delta_j^L = \frac{\delta E}{\delta a_j^L} f'(z_j^L) \tag{2.13}$$

In general, the error for each layer, l is given by equation (2.14):

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \circ f'(z^l) \tag{2.14}$$

where d^l is a vector of errors for each node in layer l.

The derivative of the loss function is given by equation (2.15), and the derivative of the activation function is expressed by equation (2.16).

$$\frac{\delta MAE}{\delta y_{pred,i}} = \begin{cases} +1, & \text{if } y_{pred,i} > y_i \\ -1, & \text{otherwise} \end{cases} \tag{2.15}$$

$$\frac{\delta f}{\delta z^l} = \begin{cases} 1, & \text{if } z^l \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.16}$$

With the Adam optimizer, two terms are used to update the weights. The algorithm calculates the moving average for both the gradient and the squared gradient of the loss function with respect to each weight. The parameters β_1 and β_2 control the decay rate of the moving averages. These are set to a default value of 0.9 and 0.999 respectively.

The momentum, m_t is then found by equation (2.17), and the variance of the gradient by (2.18).

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.17}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.18}$$

These are biased towards zero, and are corrected in equations (2.19) and (2.20)

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.19}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.20}$$

The weights are then updated by equation (2.21)

$$w_{t+1} = w_t - \frac{\mu}{\sqrt{\hat{v}} + \epsilon}\hat{m}_t \tag{2.21}$$

where ϵ is added in the denominator to avoid division by zero, ϵ is initialized to 10^{-8}, μ is the learning rate, and is set to the default value of 0.001.

This process is repeated for each batch until the whole set of training data has passed through the network. It is beneficial to run several iterations over the training data, and shuffling the data for each iteration.

The network is tested on a validation set for each iteration, to indicate how well the network is trained. Training is stopped when the loss function converges to a minimum value, and before the validation loss starts to increase due to overfitting[4].

[4]Overfitting occurs when a neural network adapts too well to the training data, thereby losing the ability to generalize.

Chapter 3

Design of experiments

In this chapter, the proposed algorithm to solve the problem of using intermittent renewable energy sources in data centers is presented. The assumptions, simplifications, and choices made while developing this algorithm are discussed. The data used for the experiments is also presented and explained, and a description of the data selection and preprocessing done for each part of the experiment is presented.

3.1 Design choices

3.1.1 Workload trace

The data trace used in the experiments performed in this thesis is from a cluster of 12 000 machines at one of Googles data centers in Georgia, USA [23].

This workload trace was chosen because it was one of the few detailed workload traces available for research. Other workloads used in relevant literature, either are not available, or they only consist of one type of load and are also sampled over a shorter period than the Google trace. Other traces found online were not chosen due to their creation dates, which in many cases were before the year 2000. Since there has been a lot of development both in hardware and software since then, it was decided to focus on a more recent workload trace.

Although the trace is highly obscured and there is no way to know what the individual jobs and tasks consist of, the trace includes information about latency sensitivity and priority, which can be used to divide the workload into separate categories. This is essential for the work in this thesis.

The chosen trace contains information about the machines, jobs and tasks for a one month period in May 2011. These are organized in different tables:

A Machine events (added / removed / updated)

B Machine attributes

C Job events

D Task events (submitted / scheduled / finished / ++)

E Task constraints

F Task usage

For the purpose of this work, the relevant tables are A, D, and F. A is used to choose a set of 100 machines, D is used to get the individual tasks for the workload trace, and F is used to gather the run data from these tasks. None of the other tables are used in this work. The machine events table (A) describes the events for the machines in the trace. The possible events are added, removed, and updated. Machines are added when they are made available to do work, removed for instance for maintenance or taken offline for other reasons, and updated if the available resources on the machines changed. The majority of the machines only have a single entry: added. The usage sample is taken from machines that only have this entry.

The tasks life cycle is described in the task events table (D). There are 9 different event types as seen in figure 3.1.

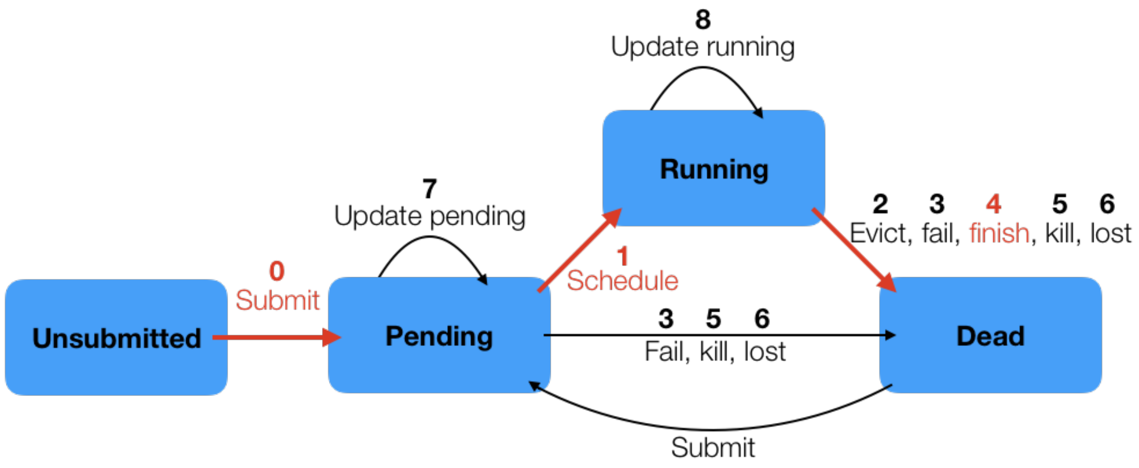

Figure 3.1: Trace state transitions, adapted from [24]. The red path is the optimal task life cycle.

The most interesting for this work, are the tasks that have run optimally. These tasks have entries in the task events table consisting of submitted (0), scheduled (1) and finished (4), illustrated with the red path in figure 3.1. For task selection, update events (7, 8) are also allowed.

Information about the tasks resource usage can be found in the task usage table (F). When a task is scheduled on a machine, information about memory and CPU usage is logged in 5-minute intervals for the duration of the task. For each interval, mean usage, maximum usage, and sampled usage is present. The sampled usage is an average from a random second of the interval. Even though the trace provides information about memory usage as well as CPU usage, in this work, CPU usage is the only usage trace that is considered.

3.1.2 Assumptions and simplifications

The algorithms presented in this work, are based on a very simplified version of reality. For instance, only 100 machines are taken into consideration when current data centers often have clusters of many thousands. The choice to narrow the data down to 100 machines was made to make the amount of data feasible to handle, and although this could impact the validity of the results, it should be enough to get a proof of concept. The assumptions made are presented in this section.

Power calculations

The power calculations are based solely on CPU usage and do not take memory usage, network connections or other factors into account. This is due to the power consumption of a server being complex, but for simplicity, it can be represented as a linear relationship of CPU utilization [25].

The machine hardware is not available from the data content. Everything in the trace is normalized based on the highest value in the whole trace. This means 0.5 CPU capacity on a machine is half of the machine in the cluster with most CPU capacity.

In this work, it is assumed that the power consumption, P, of the servers can be estimated from CPU utilization by equation 3.1 [25],

$$P_{server} = P_{idle} + u \cdot (P_{full} - P_{idle}) \tag{3.1}$$

where u represents CPU utilization of the server.

Due to the limited information about the machines used to get the trace data, in the simulations, we assume that all the 100 machines have the same hardware, and set the maximum CPU utilization of all machines to 1 (to match the maximum of the normalized values in the data trace). To calculate the power, P_{idle} is set to 300 watts, and P_{full} to 500 watts. These estimates assume that the servers are of the mid-range type.

The estimate is based on Koomeys survey from 2007 where he found the mid-range server with the highest market share in 2005 had a typical power of 495 W [26]. The typical draw is then divided by 0.66 to get a maximum power draw of 750 W [26]. To adjust for the efficiency increase in hardware from 2005 to 2011, the maximum power draw is set to an educated guess of 500 W.

Idle servers consume a lot of power, even if they are not doing anything [11]. The power consumption of idle machines is then set to 60% of the maximum power draw [11], at 300 watts.

Based on information about Power Usage Effectiveness (PUE) (see equation (2.1) in section 2.1.1) from Google's annual and quarterly reports [27], the power consumption of the center is calculated from equation 3.2

$$P_{TOT} = P_{IT} \cdot PUE \tag{3.2}$$

where $P_{IT} = \sum_{i=1}^{N} P_{server,i}$ is the total consumed power for all N machines, and the $PUE$is 2.

To deal with the high power consumption of idle machines, a secondary low power state is introduced, that will give the servers running no tasks the option to "shut down," while still being available. This is a secondary idle state, where the server is mostly shut down, but can wake up quickly if new processing tasks arrive [11]. In the "nap"-state, the servers consume 10W [11].

Placement of tasks on machines

The high latency sensitive tasks represent the interactive workload, while the lower latency sensitive tasks represent the batch and HPC workloads.

The high latency sensitive tasks are guaranteed to be placed at any time. These are never deferred, except if all 100 machines should be running at max capacity (this never happens in any of the simulations). We assume all machines have a maximum capacity of 1 and limit the placement of tasks to machines that will not exceed the capacity at any time of the task duration. It might be beneficial to set the maximum value slightly lower, at about 0.8, to avoid exceeding the capacity at any time, for a workload scheduler that is not clairvoyant[1]. In the experiments with predicted CPU usage and duration, the lower maximum limit is set.

In this work, no other considerations than CPU usage are taken into account when placing a task on a machine, this is done for simplicity, and because the power calculations are based solely on CPU usage. In reality, memory usage and hardware or software dependencies might influence the placement of a task. The advanced workings of placing tasks on machines based on different dependencies are out of the scope of this thesis.

3.1.3 SVM regression for wind power prediction

Developing methods for wind power prediction is out of the scope of this thesis, but it is included for completeness.

The wind power predictions are made using Support Vector Machine (SVM) for regression (SVR). SVR is a supervised machine learning method. The input features to the regression model are current wind speed and power output, and the SVR is trained on data from 11 months back in time.

From this, power predictions for 10 minutes to 3 hours ahead is made, in 10-minute intervals.

The SVR takes current power production and observed wind energy and uses this to estimate the prediction ahead in time. Separate models are used for different time steps ahead.

Data was provided by the master thesis work of Rune Skogstø Bryne, as a part of the collaborative project INTEGRARE.

3.2 Data description

3.2.1 Google cluster-usage traces

In this work, two different usage traces are examined. One is from a set of 100 machines, picked at random from the set of machines that only had one entry in

[1]A clairvoyant scheduler knows a tasks usage and duration in advance before it places the task

the machine events table (see section 3.1.1). This trace is named the *unaltered trace*.

The second trace is based on the number of scheduled tasks in the unaltered trace. The same number of unique tasks were selected, but none of them were killed, failed or lost at any time. We call this trace the *custom trace*.

The trace consists of several data files, split into information about the machine resources, incoming job requests, and task requests, as described in section 3.1.1. Each job consists of one or more tasks, where each task represents a process run on a machine. For each task, there is information about requested resources and sampled resource usage for each 5-minute interval. The jobs have a given priority and scheduling class. The priority ranges from 0 to 11, where higher numbers represent higher priority. The scheduling class (0 to 3) describes the job-latency sensitivity. Higher numbers represent higher latency sensitivity.

In the experiments, these numbers are used to split the workload into three different classes (as described in section 2.1.2), based on latency sensitivity, and sorted by priority.

3.2.2 Data selection and preprocessing

To get the unaltered trace, all machines that were not up and running for the entire duration of the trace were removed from the selection. Then a random set of 100 machine IDs were chosen from the remaining machines.

Information about on which machine a task was placed on can be found both in the task events (D), and task usage tables (F). The trace was selected by getting the tasks placed on machines found in the set of 100.

The trace data contain a lot of information about the workload. Both requested and actual CPU utilization of each task is logged, as shown in tables 3.1 and 3.2. On arrival, each task is logged in the task events table (D), as submitted. The tasks also have an entry in the task events table for when it is scheduled on a machine (1), and when it finished (4). There is also a possibility that the task is evicted (2), failed (3), killed (5), lost (6), or updated (7/8) during the trace period.

Table 3.1: A sample of the task events table (D)

Timestamp	Job ID	Machine ID	Task index	Scheduling class	Priority	Event type
2011-05-01 18:50:00	4765556460	1.303661e+06	1265	3	9	1
2011-05-01 18:50:00	4665896876	1.429144e+09	315	3	9	1
2011-05-01 18:50:00	4665896876	7.716048e+06	392	3	9	1

Table 3.2: A sample of the task usage table (F)

Start time	End time	Job ID	Machine ID	Task index	Mean CPU usage rate
2011-05-01 19:00:00	2011-05-01 19:05:00	6590386	711355	1	0.000332
2011-05-01 19:00:00	2011-05-01 19:05:00	6221861800	4469181033	14132	0.000202
2011-05-01 19:00:00	2011-05-01 19:05:00	6221861800	257498534	14365	0.000206

In the simulations, the mean utilization is used (table 3.2), and it was assumed that the task have this utilization for the whole 5-minute period.

To get the custom trace, all tasks with an entry in the task events table other than submit (0), scheduled (1), finished (4) or updated (7/8), were not considered. From those, a subset of the same number of tasks as in the unaltered trace was chosen at random.

3.2.3 Workload statistics

In table 3.3, a description of task duration for different scheduling classes are presented. We see that they have the same extreme values, but the mean and the median differs. For the most latency sensitive task, the mean duration is the greatest, but the median is much lower, which indicates that a lot of the tasks have a very low duration. This is supported by the 25% percentile being low as well. This group represents the interactive workload, although it has some tasks running for a longer period. These tasks could be critical systems for web services to work at all times, but since there is no information in the trace about the details of the tasks, this can only be speculation. The only metric to go by is the latency sensitivity. Of the long-running tasks in the high latency sensitive group, the higher priorities are the ones with the longest duration. The tasks with a priority less than nine, all have a mean duration of less than a day, while priorities 9, 10 and 11 have a mean duration of 21, 23 and 28 days respectively. These tasks represent about 15% of the total tasks in the scheduling class. This pattern is also true for the other scheduling classes, but with an even lower percentage of higher priorities, respectively 0.23% and 1.37% for scheduling class 0 and 1.

Table 3.3: Duration of tasks of different scheduling classes, statistics gathered from 150 000 random samples of each group.

	Scheduling class 2/3	**Scheduling class 1**	**Scheduling class 0**
Mean	3 days 09:34:00	01:27:22	00:26:45
STD	8 days 21:54:13	18:40:11	02:55:28
Min	00:00:01	00:00:01	00:00:01
25%	00:01:39	00:02:44	00:02:55
50%	00:05:21	00:09:57	00:06:55
75%	00:39:14	00:27:22	00:19:32
Max	28 days 23:53:00	28 days 23:51:15	28 days 23:41:33

In this workload, scheduling class 1 is also handled as a latency sensitive group and will be guaranteed to run at all times. This is the group that represents the batch workload.

Scheduling class 0 represents our HPC workload. Even though it consists of many tasks with a low duration, it fits the usage pattern of this load, since HPC workloads consist of jobs with many smaller tasks that can run in parallel, and are not dependent of each other. Because the jobs can be paused and started, the latency sensitivity of these jobs is low.

In the custom trace, there are 294036 unique tasks in scheduling class 0. This is the majority of the tasks. There are 56232 unique tasks in scheduling class 1, and 4016 unique tasks in scheduling class 2 and 3 combined.

Since these tasks were selected randomly from all tasks that ran non-disturbed, they reflect the whole population of non-disturbed tasks, this can be seen from table 3.4.

Table 3.4: Distributions of tasks in different scheduling classes.

Scheduling class	**All tasks**	**All non-disturbed tasks**	**Custom tasks**
0	83.79%	82.91%	82.99%
1	13.50%	15.90%	15.87%
2+3	2.71%	1.12%	1.13%

3.2.4 Wind data from a Norwegian wind farm

The wind data used in this thesis is sampled in 10-minute intervals, from a wind farm in Norway. Only one wind turbine is used for the trace because this is more than sufficient to run the 100 machines. The wind power production for the duration of one month can be seen in figure 3.2. The data presented in this figure is originally from May 2016, but to fit the workload data, it has been shifted to 2011.

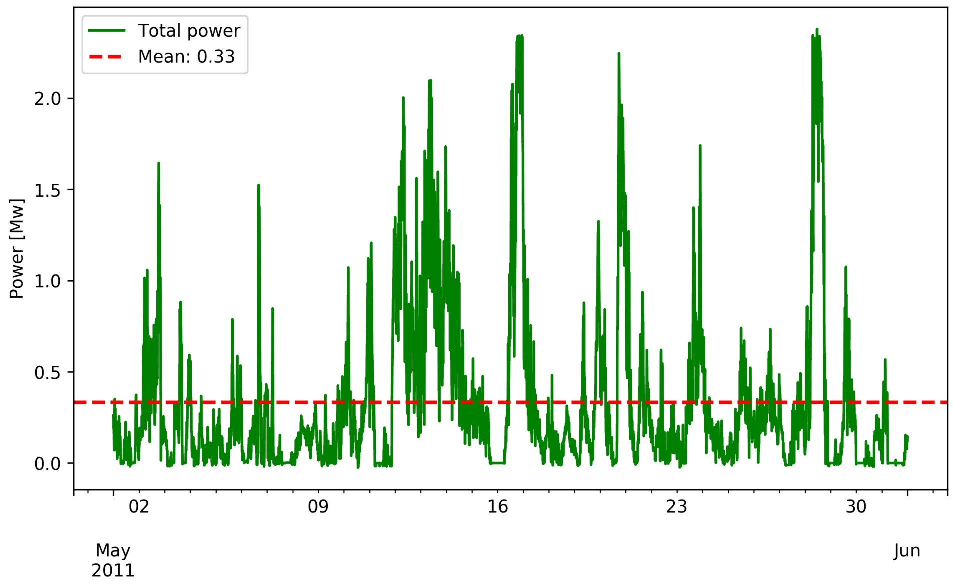

Figure 3.2: Power production for the duration of the trace.

Renewable machine availability

The power prediction is given in 10-minute intervals, as described in section 3.1.3. From the predicted power, an estimation on how many machines can run on this power is calculated by dividing the total predicted power by the maximum machine power consumption, as in equation (3.3) below. PUE and P_{server} is calculated according to equations (2.1) and (3.1).

$$Machines_{renewable} = \frac{P_{available}}{max(P_{server}) \cdot PUE} \tag{3.3}$$

Figure 3.3 shows the resulting number of renewable machines, capped at 100.

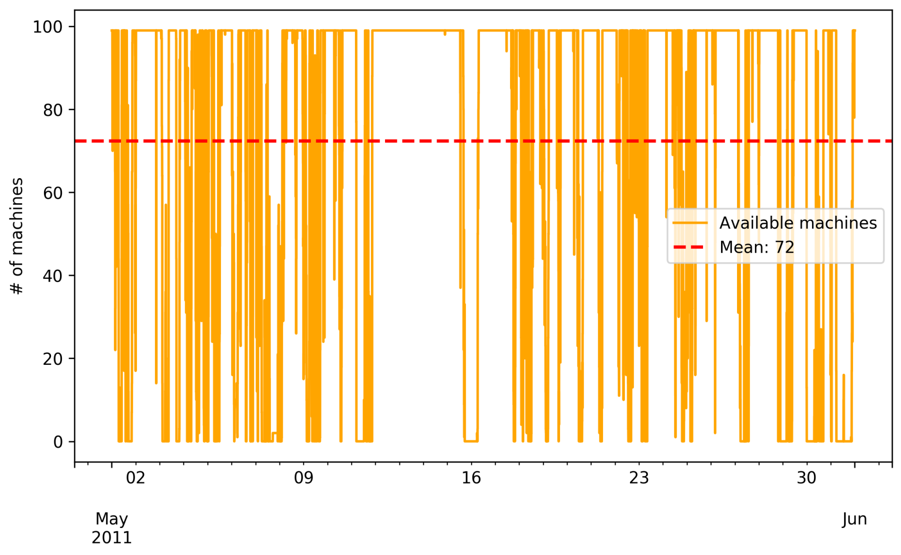

Figure 3.3: Available machines for the duration of the trace (capped at 100).

Some wind energy is available 86% of the time for the duration of the trace. 66% of the time there is enough wind energy to fully power all 100 machines at full utilization. 75% of the time there is enough wind energy to fully power half of the machines.

3.3 Description of the proposed algorithm

The goal of the model is to use as much renewable energy as possible, while still satisfying the needs of the customers. This means that the tasks marked with a high latency sensitivity always will run on arrival, while the tasks that are less time-sensitive, are free to be deferred to a later time.

3.3.1 Baseline algorithm, assuming clairvoyance

The baseline algorithm (algorithm 1) is simple. Two tables are used. First, the task events table containing the request information for each task, including the time, is split into 5-minute intervals. Figure 3.4 illustrates the process: In step 1, all the task requests are sorted into groups of different latency sensitivity. High scheduling class is linked to high latency sensitivity.

The tasks are split into three groups, high (2 and 3), medium (1) and low (0), and sorted based on priority (0-11). The priority is not directly linked to latency sensitivity, and the range of priorities is the same for all scheduling classes. The lower priorities are so-called free priorities, meaning that these tasks might not generate any revenue, and so in the original trace from Google, they can be evicted to release resources for higher priority tasks. This relationship is kept by sorting the tasks based on priority so that the higher priority tasks are placed first.

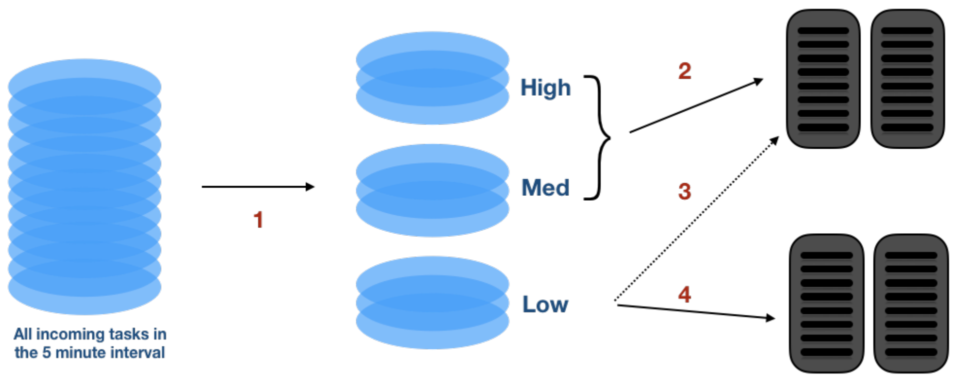

Figure 3.4: Baseline algorithm illustration. The tasks are placed on machines based on latency sensitivity.

The group of high latency sensitive tasks is placed "first" on the machines (step 2). The tasks are placed using a first-fit bin-packing algorithm [28]. From the task events table, the information about job ID and task index is used to extract each tasks run-trace from the task usage table. The run-trace is then tested against the first machine, as in figure 3.5a, to make sure there is room for the task on the machine, for all 5-minute intervals of the task run. If the machine capacity is exceeded at any time, the algorithm will try to place this task on the next machine. If the task does not fit on any machine, the task is deferred to a later period as in figure 3.5b.

In this algorithm clairvoyance is assumed, i.e. the model already know exactly how much resources a task will use at all intervals before it is placed on a machine.

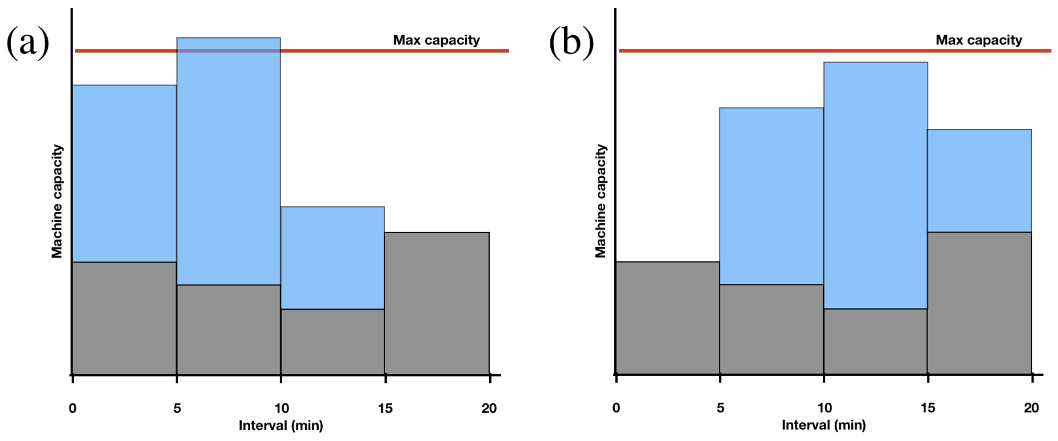

Figure 3.5: If the task (blue) does not fit at a machine (a), it either placed on another machine, or it is postponed to the next interval (b).

Obviously, this is not the case in reality. This algorithm only looks at CPU capacity. Memory resource usage and requests are ignored. This could also be something that will change the placement of tasks in real life applications.

After the placement of the high latency-sensitive tasks, it is the turn of medium tasks (step 2), and at last, the low tasks. The algorithm first attempts to place the tasks on machines that are already running other tasks (step 3). If there is no room for the task on those machines, the task is placed on a new machine (step 4). If there are no machines available with the capacity to run the task at the given interval, the task is postponed to the next time interval.

In the baseline algorithm, no power consideration is taken into account.

3.3.2 Baseline algorithm with power, assuming clairvoyance

This algorithm is similar to the baseline algorithm in the above section, the only difference is that the maximum number of machines for the low scheduling class tasks is determined based on predicted available energy. If there are fewer machines available than the previous time period, all the low tasks will be deferred, with no attempt to place any of the low latency-sensitive tasks. If the same or more machines are available, an attempt to place the tasks will be made.

[2]max machines is 100 for the baseline without power consideration for all tasks. When power is taken into account, max machines for the placement of low priority tasks is determined by the availability of machines (maximum of the number of machines running high tasks, and the number of machines that can run on predicted renewable energy 3 hours ahead). For the high priority tasks, max machines is always 100.

Algorithm 1: Baseline algorithm

```
Data: Task events table, task usage table, empty machine utilization table
Result: machine utilization table after placed tasks
create empty task tables to store deferred tasks;
start time = start of measurement period;
end time = end of measurement period;
while not at end time do
    time = start time;
    tasks = take 5 minutes of task events table;
    sort tasks into high, med, low using sorting tasks;
    append sorted tasks to previously deferred tasks;
    for task in high do
        place task on machine using place task;
    end
    for task in med do
        place task on machine using place task;
    end
    for task in low do
        place task on machine using place task;
    end
    time += 5 minutes
end
```

Algorithm 2: Sorting tasks

```
Data: Task events table
Result: Three tables with sorted tasks
high = tasks events with scheduling class 3 or 2;
med = tasks events with scheduling class 1;
low = tasks events with scheduling class 0;
sort high, med and low based on priority;
return high, med low
```

Algorithm 3: Place task

```
Data: Task events table, task usage table, machine utilization table,
      previously deferred tasks
Result: machine utilization table with task placed, list of deferred tasks
usage = get task CPU utilization from usage table;
if usage duration is not none then
    while machine <= max machines[2] do
        machine status = machine utilization for duration of task usage;
        if machine status + task usage > 1 for any intervals then
            if machine = max machine then
                defer task;
            end
        else
            place task on this machine;
        end
    end
end
```

When placing individual tasks, power considerations are also taken into account. To make sure there is enough capacity to run the whole task, the algorithm checks the predicted available machines for all intervals of the task run. If there are not enough machines at some point of the duration of the task, the individual task is deferred to the next timeslot.

Since the machines that run the higher latency sensitive tasks are guaranteed to run at any time, lower latency sensitive tasks have the option to be placed on these machines.

In the baseline algorithm considering power, the algorithm is clairvoyant with respect to task usage, but not energy production. The available renewable power is given by prediction 3 hour ahead. To get the number of available machines, the algorithm picks the highest number of the minimum number of predicted machines 3 hours ahead and the minimum number of available machines running high latency sensitive tasks for the duration of the task. As mentioned earlier, the clairvoyant model is not a feasible approach in reality, due to the task usage not being known in advance.

3.3.3 Non clairvoyant algorithm with task usage prediction

To deal with the clairvoyance, a neural network is used to predict the tasks mean CPU usage and duration. The non-clairvoyant algorithm bases the placement of tasks on the predicted CPU value and duration of the task, and the predicted power production three hours ahead. Other than this change, the algorithm works in the same way as algorithm 1. The placement of tasks based on predicted values is described in algorithm 4.

Algorithm 4: Place task (based on predicted usage)

```
Data: Task event, machine utilization table, predicted machine utilization
      table
Result: machine utilization tables with task placed, list of deferred tasks
1  task ID = get task ID;
2  usage = get task CPU utilization from usage table;
3  predicted = predict task CPU usage and duration;
4  if predicted duration is not none then
5      while machine <= max machines[a] do
6          predicted machine status = predicted machine utilization for
           predicted duration;
7          current machine status = current usage of machine;
8          if predicted machine status + predicted task usage > 1 for any intervals
           or current machine status + predicted task usage > 0.8 then
9              if machine = max machine then
10                 defer task[b];
11             end
12         else
13             place task on this machine;
14         end
15     end
16 end
```

[a]max machines is the greater value of the minimum number of machines running high latency sensitive tasks for the duration of the task, or the minimum number of renewable machines available for the duration of the task. If the duration is more than three hourss, only the first three are taken into consideration due to the prediction only being three hours ahead.

[b]For high priority tasks, the task will be placed on the machine with the lowest current utilization instead of being deferred.

When placing individual tasks on a machine, it is made sure that the tasks will not exceed the capacity of the machine. Since the placement now is based on the predicted mean CPU usage, there is a check to make sure that the predicted usage of the previously placed tasks on the machine, plus the tasks predicted CPU usage will exceed the machine capacity at any time of the predicted duration. An additional check is made, to see if the current real usage on the machine plus the tasks predicted CPU usage will exceed 80% of the machine capacity. This lower bound is to account for the lack of variation in the prediction, since the predicted task usage is constant, and does not take into account the varying CPU usage of a task.

3.4 Neural network for task usage prediction

In the clairvoyant algorithm, the usage profile of the task is given in advance. This is obviously not the case in real-world applications.

To place the tasks on machines the most efficient way, there is a need to know in advance how much CPU a given task would use, and how long the task will take to finish.

When a task is submitted, a lot of information is made available (see table 3.1). For example the requested CPU, the scheduling class, the priority and the event type. There is also information about the user ID and job ID, which might give some information about how the task will run.

Neural networks are good at solving regression problems and are able to replicate any function, therefore, a solution to remove the clairvoyance of the scheduler is proposed, by using a neural network to estimate a tasks CPU usage and duration. The functions and details of the neural network are described in section 2.3.1.

3.4.1 Data preparation and inputs

Two networks are developed for each scheduling class. One for the prediction of mean CPU usage rate, and one for prediction of duration.

The inputs to the network are gathered from the task events table, and consist of several values:

1. Resource request CPU

2. Priority
3. Event type
4. Scheduling class [3]
5. User ID
6. Time of day
7. Day of week
8. Job ID

The User ID is mapped from the hashed values in the original data to individual integer values.

Before the inputs are sent through the network, they are normalized. Each feature (input) is scaled to a range from 0 to 1. This is to avoid weighing one feature more than another since they differ in scale.

The transformation is done by equation 3.4, for each feature:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.4}$$

where X is the original input feature, X_{min} and X_{max} is respectively the minimum and maximum instance of all values for this feature.

To gather the training data, a selection of 150 000 tasks are chosen for each task group (low, medium, high latency sensitive) from all the task events that ran normally. The target values are gathered by finding the actual usage trace for each task from the task events table. The target CPU values for each task is calculated by taking the average of the mean CPU utilization for all 5-minute intervals of the duration of the task, and the target duration is the sum of 5-minute intervals of a task.

3.4.2 Network structure

To get the best results, separate networks are used for the different task groups. There are two networks for each group, one for prediction of duration, and one for

[3]Only in use for the high latency sensitive tasks, as there are fewer of these and they are scheduled together.

prediction of CPU usage. The networks are similar for all groups. The differences are stated below.

For task CPU usage prediction, a network with four hidden layers is used, as illustrated in figure 3.6. The layers have 7, 250, 150, 100, 50 and 1 nodes, in order from input to output, where the output is the predicted CPU usage. The number of nodes was chosen based on trial and error, with this setup giving the lowest prediction error. Each layer uses the ReLU activation function.

The input vectors x_i, consist of the scaled values from the list in the previous section.

These values are sent through the network as described in section 2.3.1.

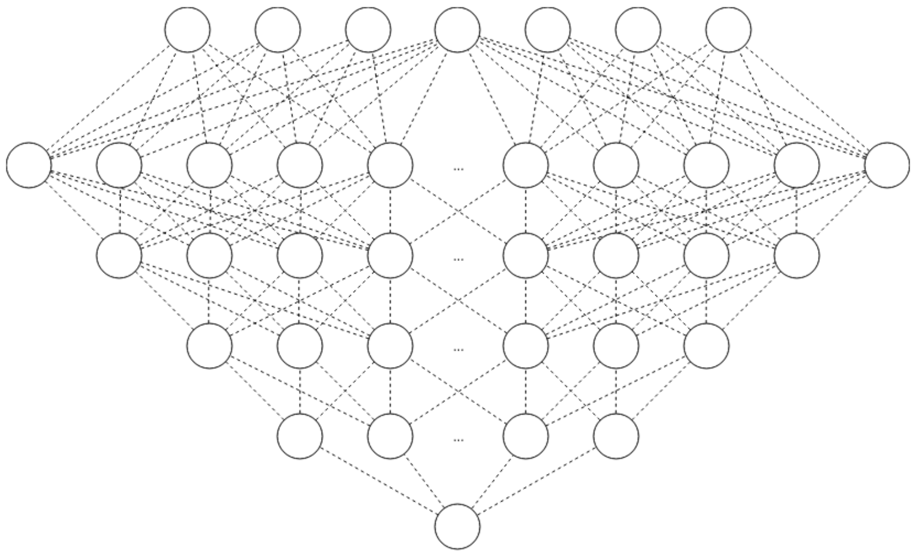

Figure 3.6: A simplified illustation of the neural network used. All networks have six layers, consisting of an input layer (top) and an output layer (bottom), and four hidden layers with 250, 150, 100 and 50 nodes from top to bottom. All nodes in the previous layer is connected to all nodes in the next.

The networks have the same structure for both CPU prediction and duration. The difference is the output value. For the duration prediction the output is the predicted number of 5-minute intervals a task will run, while for the CPU usage, the prediction is the total mean CPU usage of the task.

This prediction does not take into account that a task usually does not have the same utilization for the whole duration, and therefore is not able to reflect the variation of CPU utilization for a task.

The networks performances are evaluated by comparing the prediction error with the error of guessing at the mean of the samples and by the fit of the regression line from the predicted and real values. The mean of the predicted errors and the median are good indicators of the performance of the network.

The R^2 value is also used for evaluating the performance of the prediction model, as in equation (3.8). The R^2 value is calculated from the sum of the residual squares of the prediction 3.7, and the total sum of squares 3.5. The total sum of squares is the sum of all the observations squared distance from the overall mean.

$$SS_{tot} = \sum_i (y_i - \hat{y})^2 \tag{3.5}$$

Where y_i is the real value for sample i, and $\hat{y}$ is the mean of all real values, as in equation (3.6):

$$\hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{3.6}$$

The residual sum of squares 3.7 is the sum of the squared difference between the predicted and the actual values for all the observations.

$$SS_{res} = \sum_i (y_i - y_{pred,i})^2 \tag{3.7}$$

The R^2 value is then calculated from the residual and total sum of squares:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{3.8}$$

A good fit gives a R^2 value close to 1.

Chapter 4

Results and discussion

In this chapter, the results from the previously described experiments are presented. To start with, data from the unaltered original trace from the original 100 machines is shown.

Baseline results for comparison are presented, for both the unaltered and the custom trace. The baseline is the clairvoyant algorithm that places tasks as it is, without power consideration, see section 3.3.1.

Then, the results from the power-aware algorithm (section 3.3.2) are presented and discussed in comparison to the baseline results. The results from the non-clairvoyant algorithm with task usage prediction (section 3.3.3) is evaluated and compared to the baseline and to the power-aware algorithm. We look at the power consumption both in total and with respect to percentage renewable, CPU utilization and the number of deferred tasks, and estimate how well the algorithm performs compared to the baseline and the power-aware algorithms. The reduction of machines in use during the periods of low power availability gives a good indication of how well the algorithm adapts to power availability.

At last, the results from the neural networks are presented and evaluated.

4.1 Original data

On the original machines from the unaltered trace, the mean CPU utilization for each 5-minute interval on the individual machines range from 0.055 to 0.488 as shown in figure 4.1, but on some machines, the maximum CPU utilization exceeds 1 at certain points. Since the mean CPU usage of each task is only given in 5-

minute intervals, there is some uncertainty linked to these numbers. Even so, if the capacity is exceeded at some points, this is not critical. The low utilization is typical for data centers, and it leads to a lower efficiency than if fewer servers were in use, with higher utilization.

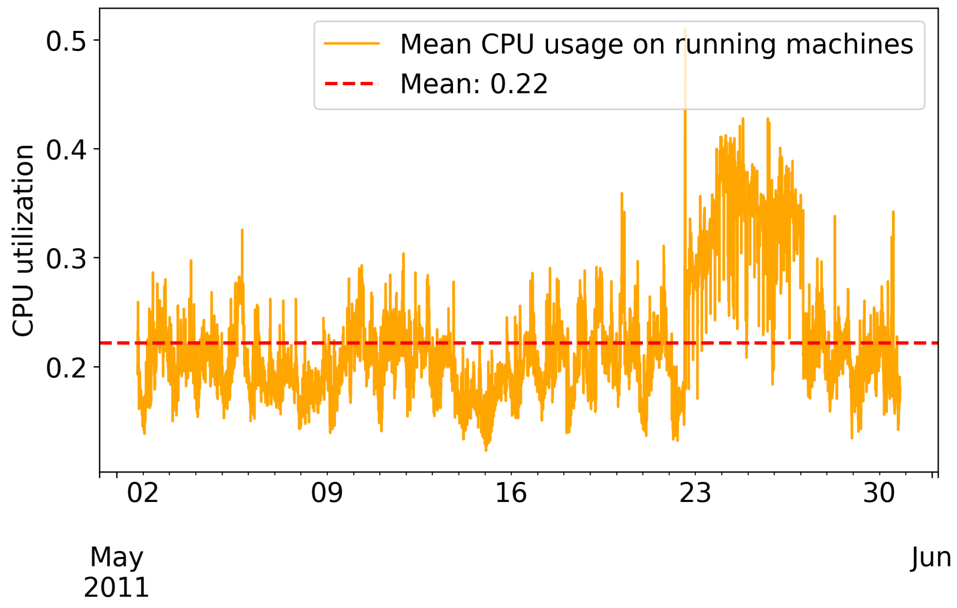

Figure 4.1: The mean CPU utilization on the running machines (in this case all 100 for most of the time). The utilization is lower than 0.3 most of the time.

We tested two different power configurations, one where all machines are turned on at any time, and another where the machines have a low power state when the machines are not doing any processing, that consumes much less power (10W) [11]. The first configuration is shown in figure 4.2. The total power consumption for the month-long trace period is 23866 kWh.

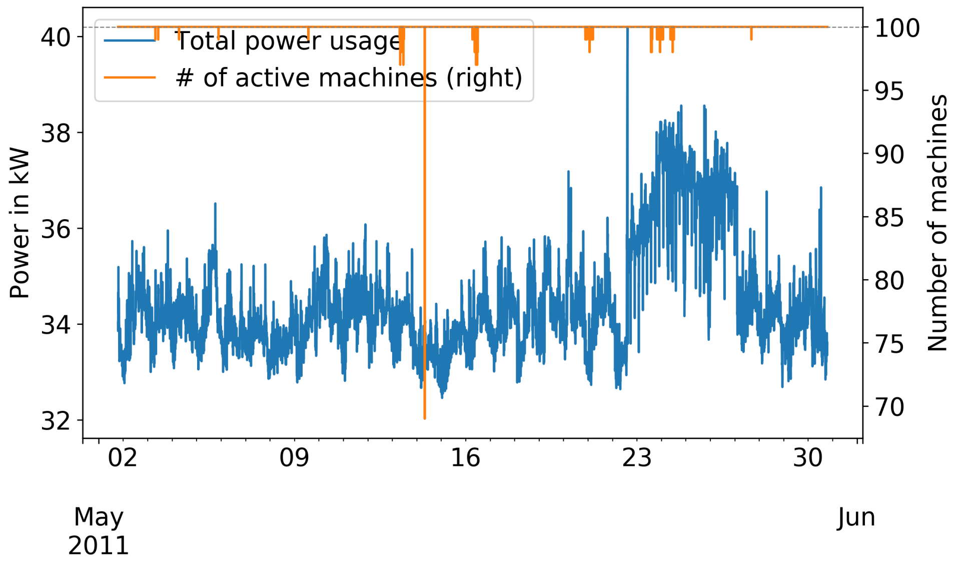

Figure 4.2: Power usage for the original 100 machines, for the month of May 2011. Total energy: 23866 kWh. The dotted line is the mean machines in use.

If we assume all machines are in a low power state until needed, we can start them up on demand, and use less power when the machines are not doing any processing. We assume the server consumes about 10W in a low power state [11]. In figure 4.3 it can be seen that the power consumption is slightly lower when the low power state is implemented, with a total power consumption of 23863 kWh.

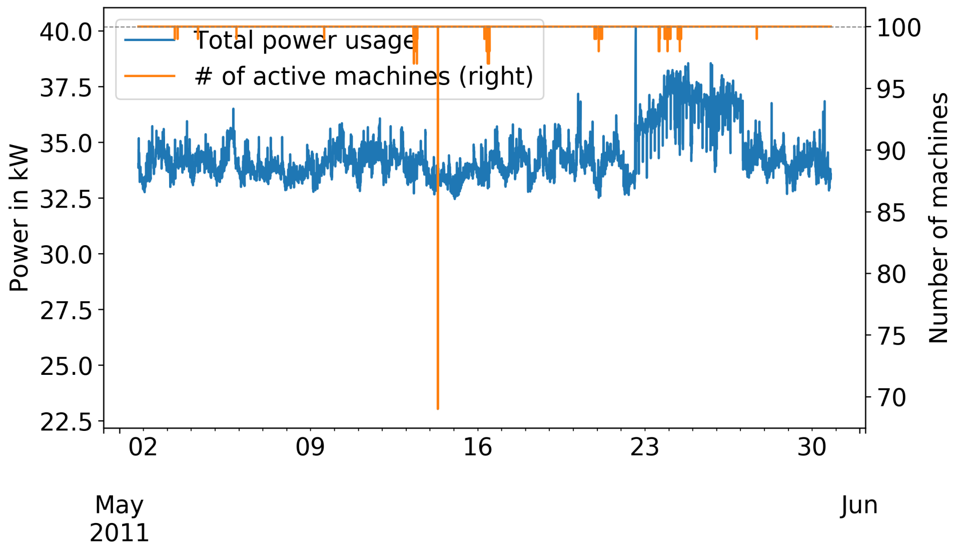

Figure 4.3: Power usage for the original 100 machines, for the month of May 2011. Here with idle machines in a low power state. Total energy: 23863 kWh. The dotted line is the mean machines in use.

The original trace does not consume significantly less power when having a low power state for the machines. This is not surprising since almost all of the machines run at any time: the mean number of active machines is 99.98. The fraction of renewable energy used is the same for both configurations, at 81% of the total power consumption.

To reduce the number of machines running at low capacity (which means low efficiency), it would be beneficial to fill up as many machines as possible, without exceeding the processing capacity of the machines. This is what the developed algorithm attempts to do.

4.2 Results from baseline algorithm

The baseline results presented in the following paragraphs are run without taking any power availability into account. The tasks are placed on machines as they arrive, regardless of available wind energy. The consumed power is calculated based on CPU utilization of each machine, with idle usage set to 300 watts, and full usage to 500 watts.

The traces are created from the baseline algorithm with clairvoyance, described in section 3.3.1, so the task profile (duration and CPU usage) is known at the time of placement.

The CPU utilization of the running machines is higher than for the original trace load since the baseline algorithm is clairvoyant and therefore can place a task on a machine with a certainty of never exceeding the machine capacity.

4.2.1 Unaltered trace

In the baseline results with the clairvoyant algorithm, we never have 100 machines in use at the same time. The average number of machines in use is 29, and the maximum number of running machines is 43, as seen in figure 4.4. This means that 57 machines are never used to process any tasks from this setup. The low power-state would be much more efficient at saving energy for this setup.

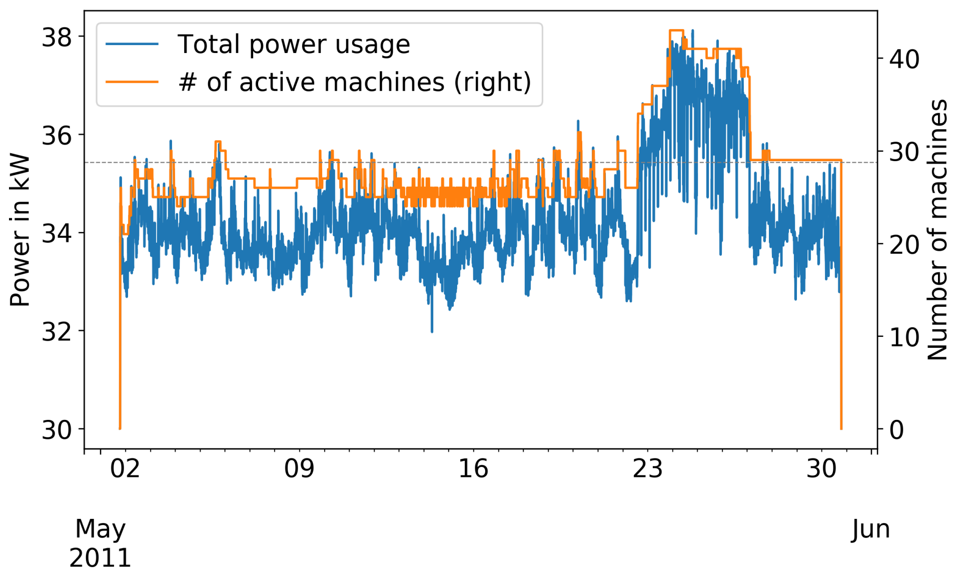

Figure 4.4: Baseline results for the unaltered trace taken from 100 machines. Total energy used: 23787 kWh. The dotted line is the mean machines in use.

In figure 4.5, the results from the same algorithm, but with a low power state introduces are shown. Comparing figure 4.4 and 4.5, there is less energy used in the low-power state configuration, with a reduction from 23787 kWh to 9457 kWh. There are many machines that are not utilized at any given time.

The renewable utilization is 84% of the total energy consumption with the low power states and 81% without. With the low power state, this is an improvement from the original trace. The overall power consumption is also lower than for the original trace, both for the configuration with and without low power state. This is as expected since many of the machines are in the low-power state at any given time.

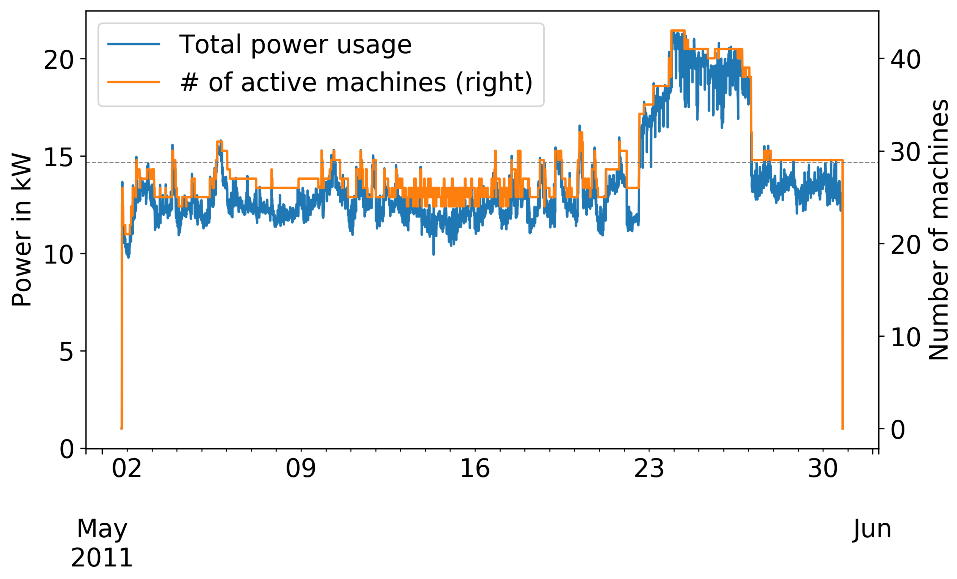

Figure 4.5: Baseline results for the unaltered trace taken from 100 machines. Machines with no tasks are in low-power state. Total energy used: 9457 kWh. The dotted line is the mean machines in use.

Figure 4.6 shows that the average utilization of the running machines is much higher than for the original trace (figure 4.1), giving us a higher processing efficiency. This efficiency increase can also be seen from the lower total energy consumption being reduced by 60%.

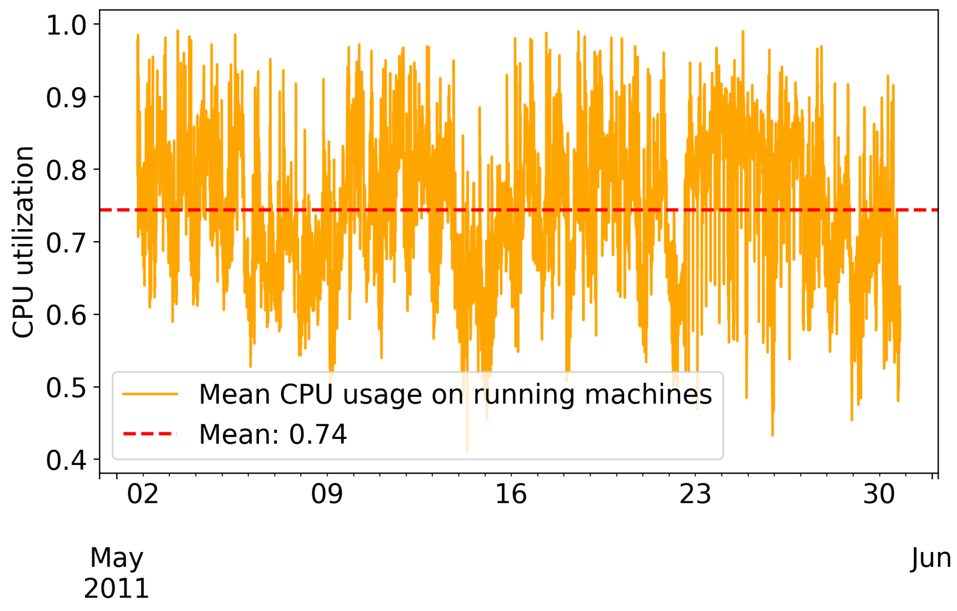

Figure 4.6: The mean CPU utilization on the running machines (ranging from 20 to 43).

4.2.2 Custom trace

In the case of the custom trace (section 3.1.1), the average number of machines used is 33, and the maximum number is 60, as shown in 4.7. This trace has a higher overall energy consumption of 24023, which is higher than the original trace loads 23787 kWh since we only place a task once. In the trace from the 100 machines, some tasks are stopped and started multiple times, so when taking the same number of scheduled tasks the number of total tasks is higher.

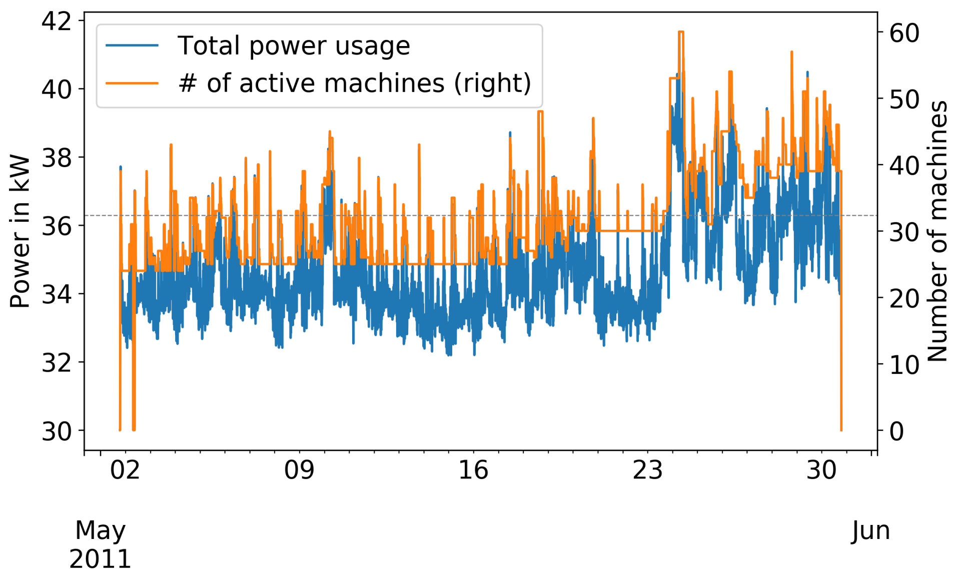

Figure 4.7: Baseline results for the custom trace. Total energy: 24023 kWh. The dotted line is the mean machines in use.

As with the unaltered trace, the energy consumption of the configuration with a low power state consumes less power than the one without, as seen in figure 4.8. The fraction of power from wind energy for the trace is 83% for the low power configuration, and 81% for the regular configuration.

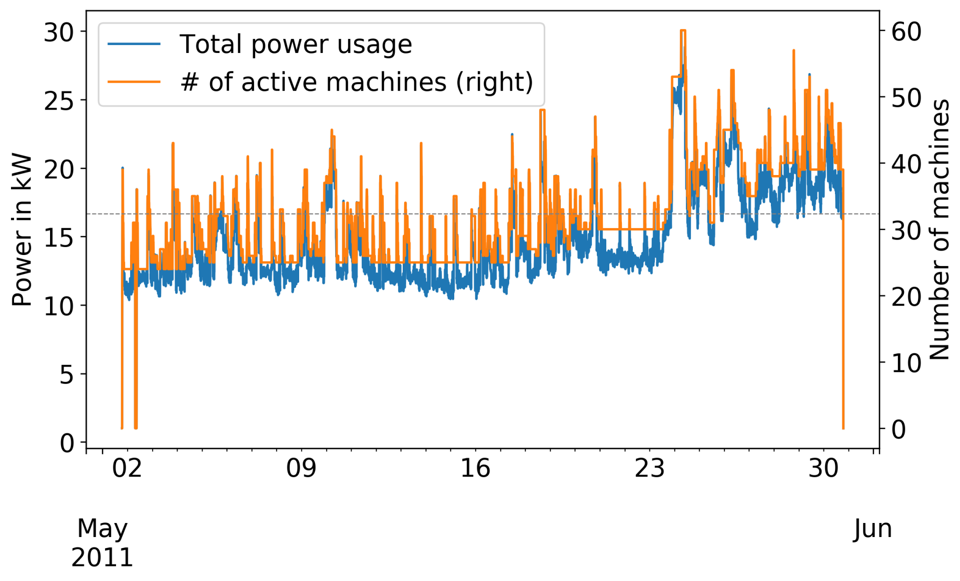

Figure 4.8: Baseline results for the custom trace. Machines with no tasks are in low-power state. Total energy used: 10413 kWh.

Similar to the results from the unaltered trace, the CPU utilization on the running machines is higher than the original. Figure 4.9 shows that the mean CPU utilization of the running machines is 0.74, a huge increase from the original mean of 0.22. We also see that the mean capacity never exceeds 1.

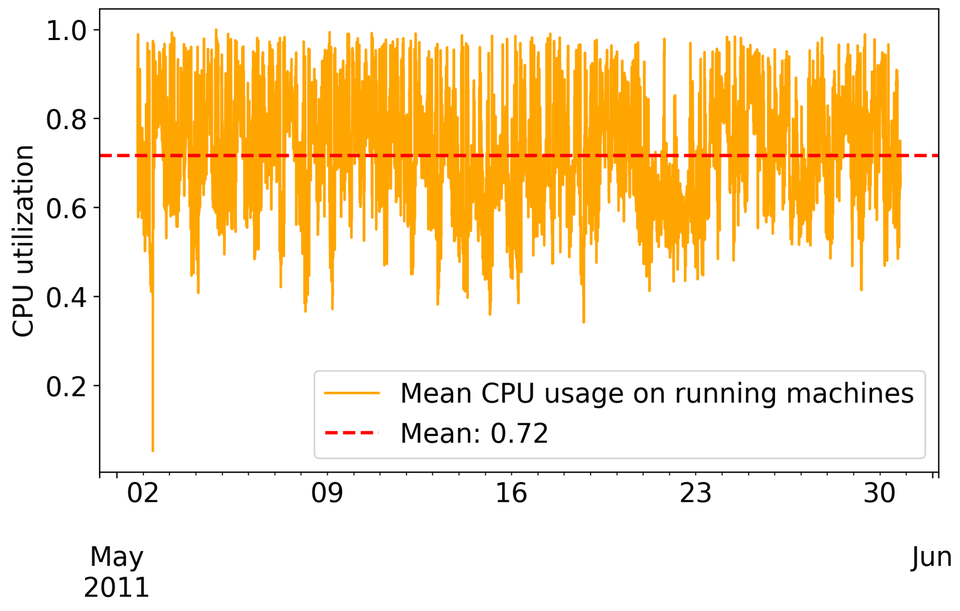

Figure 4.9: The mean CPU utilization on the running machines.

For the baseline algorithm, no tasks are deferred, since all tasks are placed at the time they arrive.

4.3 Results from power-aware algorithm

The power-aware algorithm (described in section 3.3.2), takes available renewable power into account when placing a task. The algorithm is clairvoyant in respect to the task duration and CPU usage, but not to wind power availability. It uses the predicted power availability 3 hours ahead to determine renewable machine availability. For this section, we only look at the configuration with the low-power state of machines. The pattern is the same as with the baseline algorithm, with the low power state using less overall power.

4.3.1 Unaltered trace

As in the previous results, the overall energy consumption goes down for the power-aware algorithm from 23787 kWh to 8902 kWh. In this case, the fraction of renewable power is at 85%, which is slightly better than the baseline algorithms 84%.

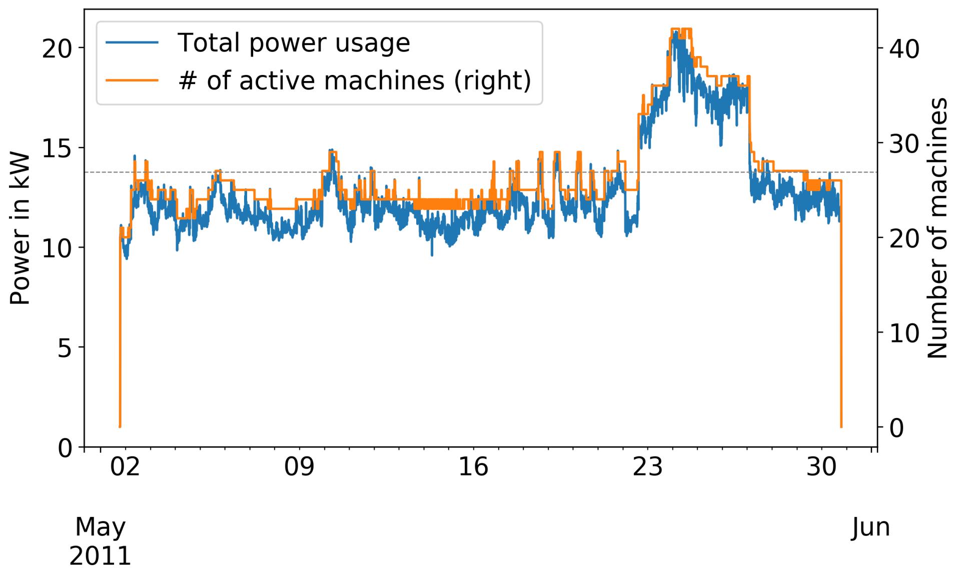

Figure 4.10: Results from the power-aware scheduler for the unaltered trace. Machines with no tasks are in low-power state. Total energy used: 8902 kWh.

The mean number of running machines is 27, and the maximum number of machines in use at any time is 42.

For 21% of the time, there is less power available than needed to run the machines. The power-aware algorithm manages to reduce the number of machines in the periods of low renewable availability 95% of the time. On average, there are 2.12 fewer machines running in these intervals than with the baseline algorithm. The maximum reduction for any interval is 8 machines.

The mean CPU utilization of the running machines of 0.75, figure 4.11, is slightly higher than the baseline algorithm at 0.74.

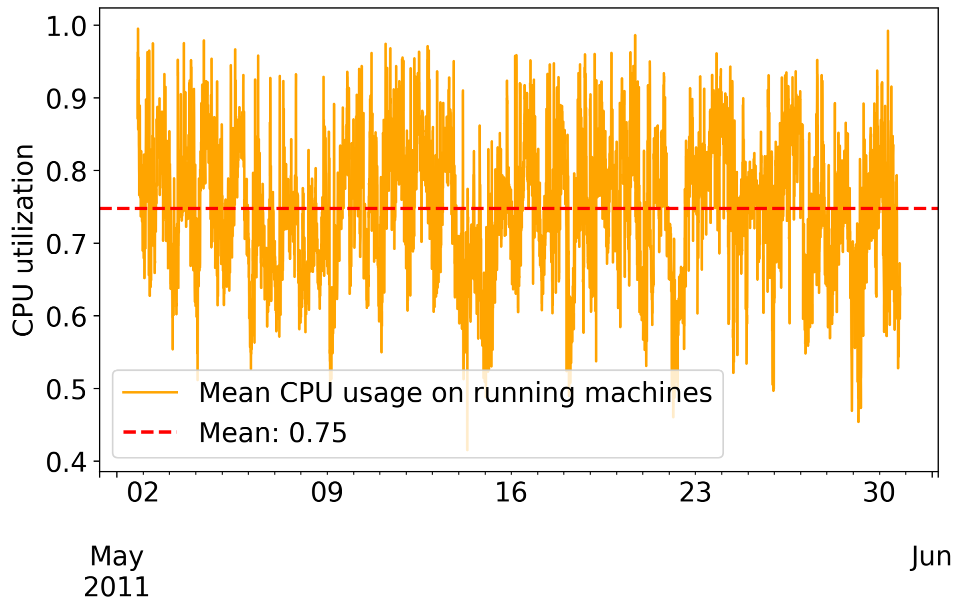

Figure 4.11: The mean CPU utilization on the running machines.

With power-awareness, a total of 524 tasks were deferred at least one interval, all of these were low priority tasks. The pattern of the deferred tasks is shown in figure 4.12. The pattern matches up with the periods of low wind power production, as we can see from figure 3.3.

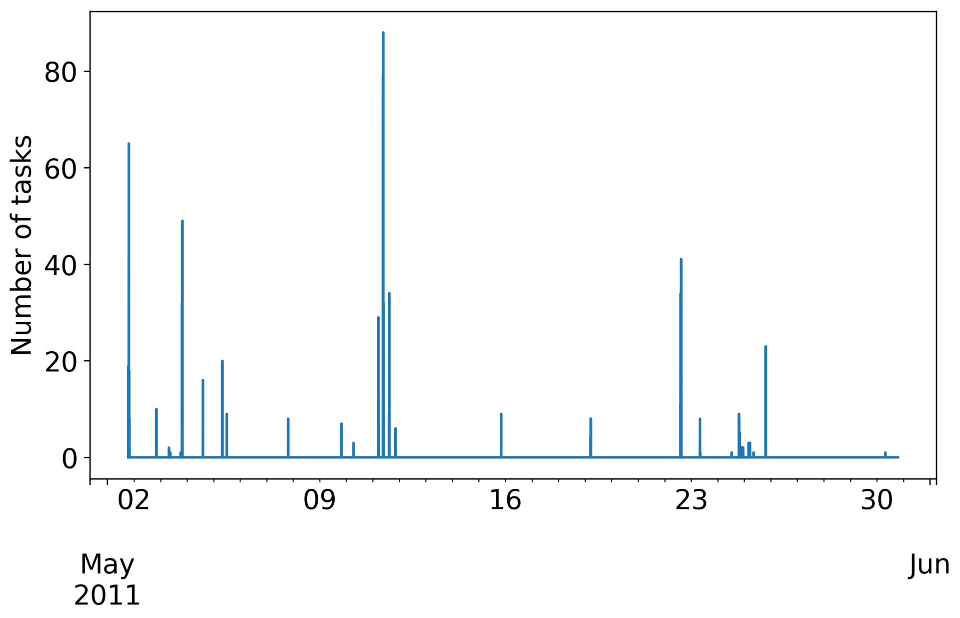

Figure 4.12: Number of deferred tasks each interval. A total of 524 tasks.

The tasks were not deferred by a lot of time, with a maximum deferral time of 35 minutes, and a mean of 8.8 minutes.

4.3.2 Custom trace

For the custom trace, chosen from the tasks that ran normally (see section 3.1.1), figure 4.13 shows that the power-aware algorithm has fewer machines running overall, and less total energy is consumed compared to the baseline algorithm, with a reduction from 10413 kWh to 10001 kWh.

The mean number of machines is 32 and the maximum number is 59.

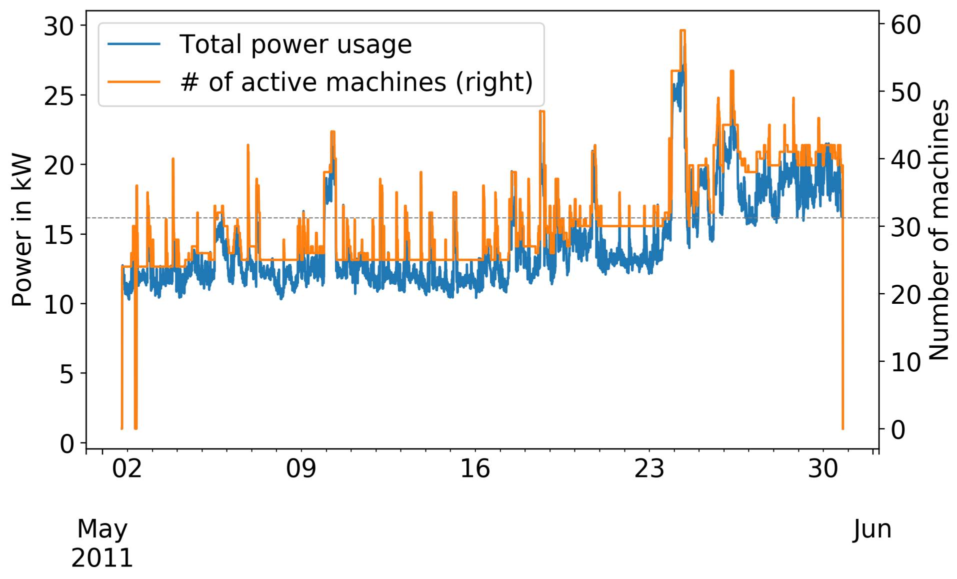

Figure 4.13: Results from the power-aware algorithm for the custom trace. Machines with no tasks are in low-power state. Total energy used: 10001 kWh.

From the total energy consumption, 84% is from wind energy, compared to the 83% in the baseline algorithm.

For 21% of the time, there was not enough renewable energy to power the running machines. For these intervals, the number of machines running compared to the baseline algorithm was reduced 55% of the time. There was a mean of 4.8 fewer machines running in these intervals compared to the baseline results, and the maximum reduction was 17 machines.

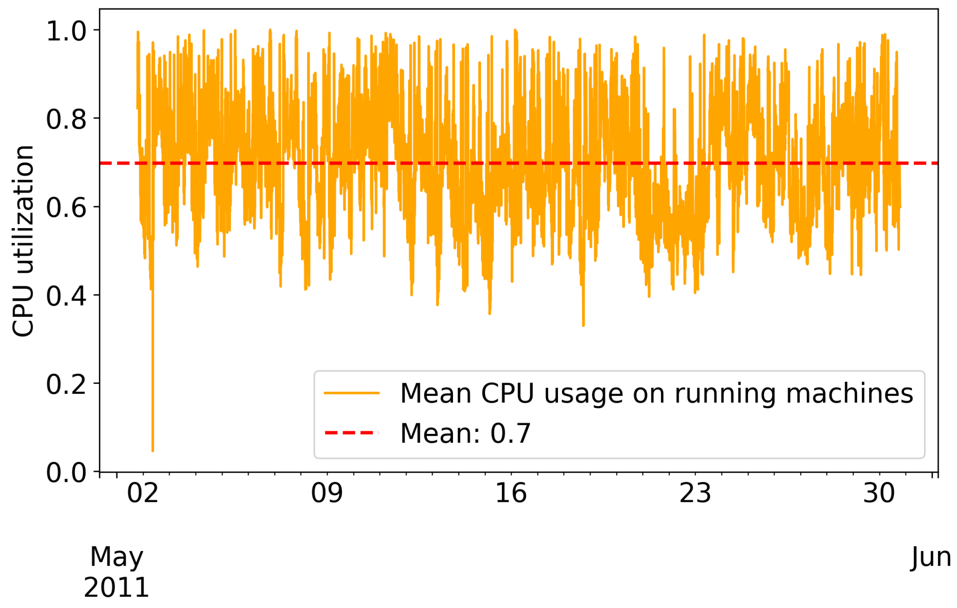

Figure 4.14: The mean CPU utilization on the running machines.

With power-awareness, a total of 4601 tasks were deferred for at least one interval, all of these were low priority tasks. As with the unaltered trace, the pattern of deferred tasks follows the renewable machine availability.

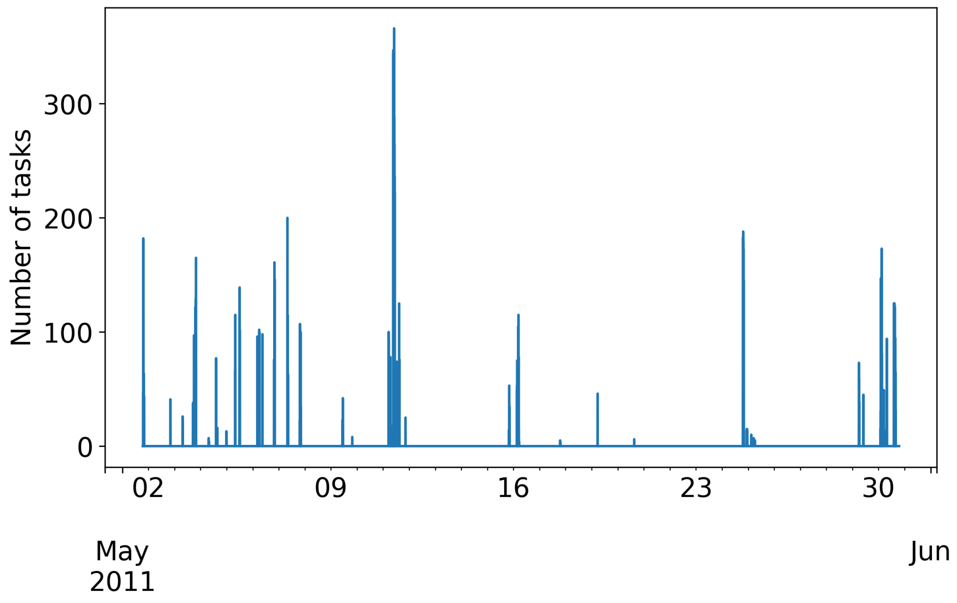

Figure 4.15: Number of deferred tasks each interval.

The mean task deferral time is 12.5 minutes, and the maximum deferral time 60 minutes.

4.4 Results from algorithm with predicted CPU usage and duration

The case of the algorithm using task CPU usage and duration prediction shows very similar results to the clairvoyant algorithm. One major difference is the number of deferred tasks. In figures 4.18 and 4.19, we compare the predicted and actual energy usage of the trace. The predicted usage of 10810 kWh is greater than the actual usage of 9738 kWh. As will be shown in section 4.6, this can be explained by the prediction errors. Either the duration of tasks can be overestimated, or the CPU usage of tasks that in reality is 0 for some or most intervals could be predicted to be much higher.

Overall, the algorithm with prediction uses less power than both the baseline results and the clairvoyant power-aware algorithm. The results are compared in table 4.2 in the next section.

In figure 4.16 and 4.17, the predicted and real CPU utilization on the running machines is presented. The mean predicted CPU utilization of 0.73 is smaller than the mean of the real utilization at 0.75, and the real utilization exceeds 1 in some intervals. This is not critical but would reduce the performance of the overloaded machine and lead to slower processing. This is not taken into account in these results, due to the complexity of calculating the impacts, and due to the simplicity of these experiments. The mean CPU utilization exceeds 1 for 3.9% of the time.

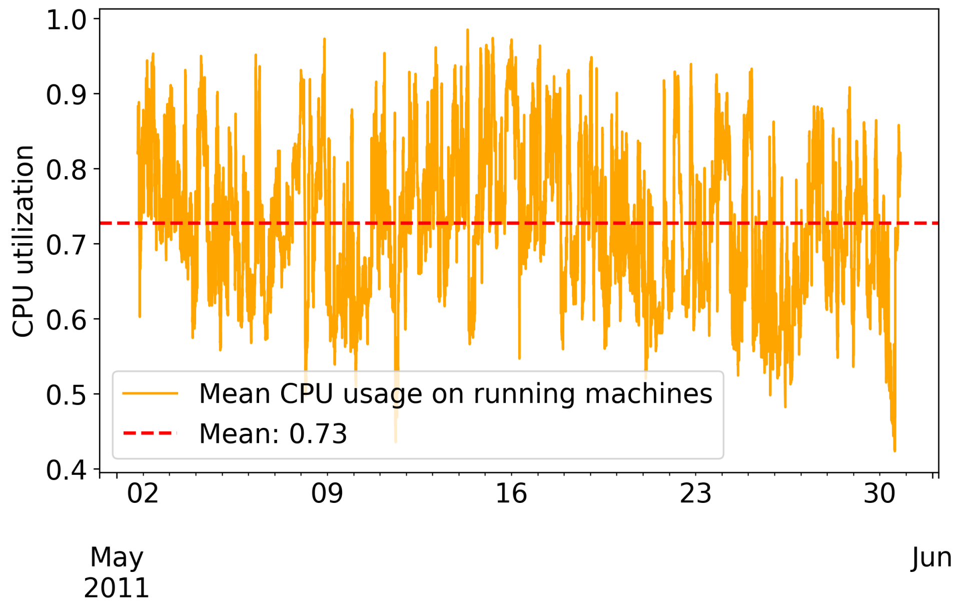

Figure 4.16: The *predicted* mean CPU utilization on the running machines.

This never happens in the clairvoyant algorithm, and it illustrates the need to predict a tasks usage not only as a static mean but as a variation over the trace runtime. These results would not be acceptable in a real data center and underline the importance of leaving some extra room on the machines for unexpected traffic.

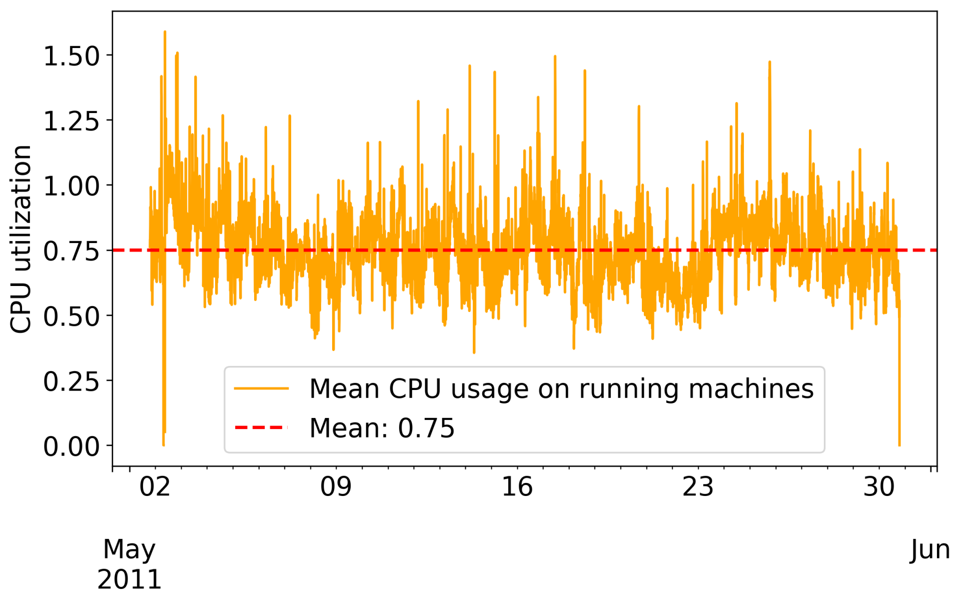

Figure 4.17: The *real* mean CPU utilization on the running machines.

In figure 4.18 the predicted power consumption of the machines are shown, and in figure 4.19 is the real power usage and machines running.

The results give 81% renewable energy for the predicted results, while for the real results the machines are run on 84% renewable energy.

We can see that the real energy consumption of 9738 kWh is lower than the predicted energy consumption of 10810 kWh. The predicted maximum number of machines running is 67 machines, and the mean number is 34 machines. In the real results, there is a maximum of 62 machines running and a mean of 30. This is lower than for the predicted.

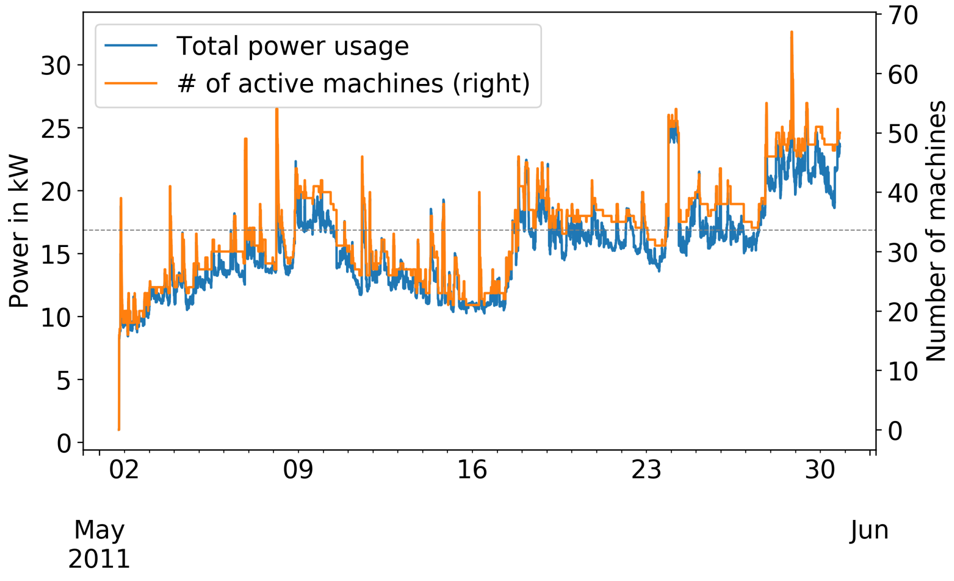

Figure 4.18: *Predicted* results for the custom trace with the prediction based algorithm. Machines with no tasks are in low-power state. Total energy used: 10810 kWh.

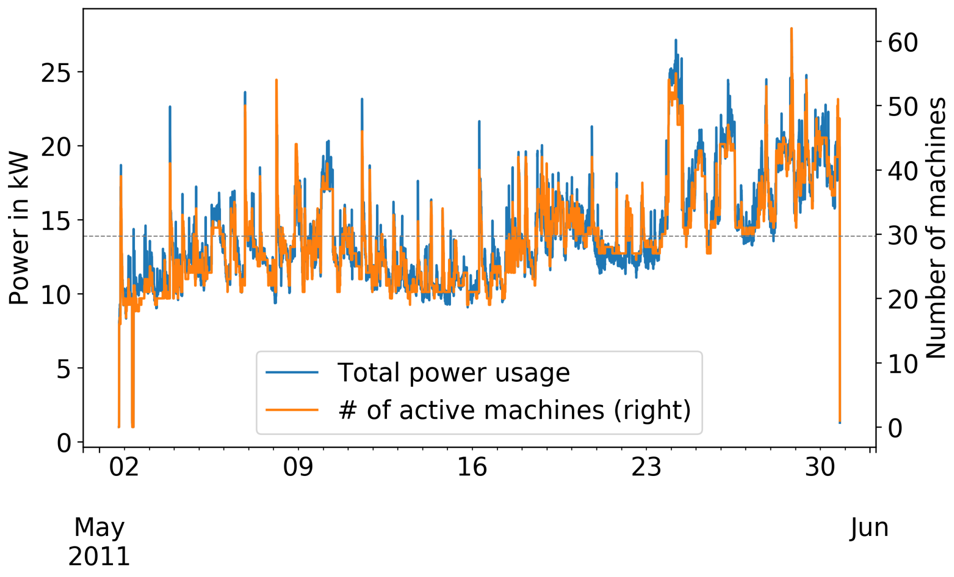

Figure 4.19: *Real* results for the custom trace with the prediction based algorithm. Machines with no tasks are in low-power state. Total energy used: 9738 kWh.

As with the previously seen results, there was enough power to run the machines 21% of the time. For 79% of this time, the number of running machines was reduced compared to the baseline algorithm for the same trace. This is a better result than for the power-aware algorithm but can be explained by the machines running on higher capacity. There was a mean of 5.6 fewer machines running, and the maximum number of reduced machines was 23.

Figure 4.20 shows the number of deferred tasks for each interval. The total number of deferred tasks is much higher than for the clairvoyant algorithm, with a total of 23230 tasks being deferred.

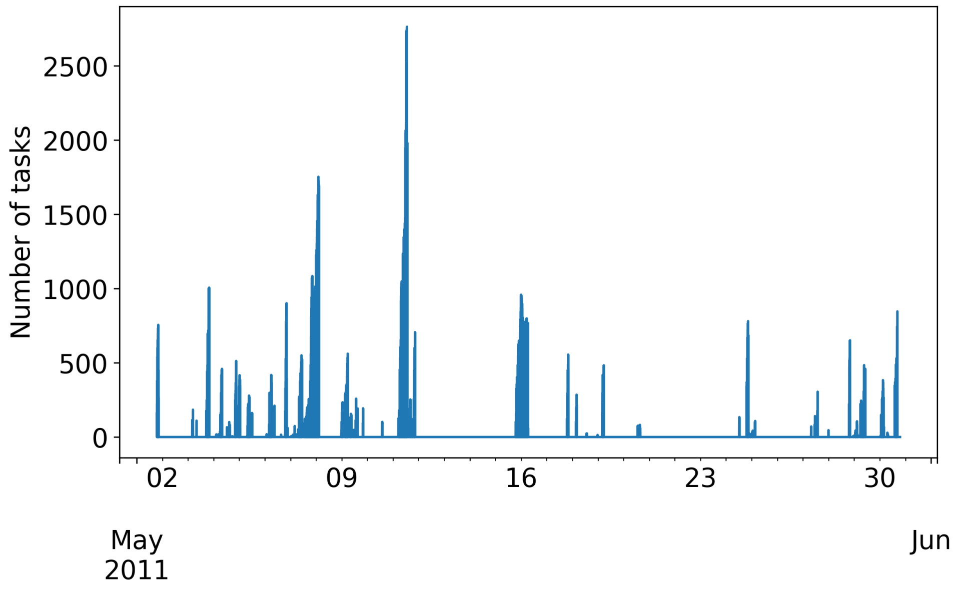

Figure 4.20: Number of deferred tasks each interval.

The pattern is the same as in the power-aware algorithm, with the deferred tasks matching up with renewable availability. The delay is greater than for the other results. The maximum deferral time is 393 minutes and the mean 76 minutes.

4.5 Comparing the results

Overall, the prediction algorithm outperforms the clairvoyant power-aware algorithm in regards to power consumption and does much better than the original trace. It does though, overload some of the machines capacities for a large fraction of the time, which means that the quality of the processing would decrease significantly. To deal with this, a larger overhead would be needed, or a more sophisticated method for prediction of task usage.

Of the 62 machines running at some point of the duration of the trace, 60 of them are oversubscribed (running over their capacity) at some time when the predicted values are used to place the tasks. Of these, the maximum time a single machine is running above its capacity is 47% of the time. The ”earlier” machines (0,1,2...)

are the ones that are spending the most time being oversubscribed. The mean is oversubscription 12% of the time and the median 6%. Of the oversubscribed machines, 75% of them are oversubscribed less than 14% of the time.

The number of oversubscribed machines can be reduced by increasing the overhead (lowering maximum capacity of machines when placing the tasks), with the risk of tasks being deferred even when there is enough power available, or by developing a more sophisticated model for predicting task usage over time.

The power-aware algorithm acts as expected, and reduces the number of running machines in the periods of low power availability, and also reduces the overall electricity consumption. For the unaltered trace, the number of running machines in these intervals was reduced 96% of the time, and for the custom trace 56% of the time with the power-aware algorithm. With prediction, for 79% of the time, there was a reduction in machines running in these intervals.

Compared to the power-aware algorithm, the prediction algorithm defers more tasks overall, and it defers them for a longer time than the clairvoyant algorithm. Still, the deferral is within acceptable bounds for the scheduling class, and none are deferred for more than 7 hours. An overview of the results from the different algorithms are presented in table 4.1 for the unaltered and the original trace, and in 4.2 for the custom trace.

Table 4.1: Overview of results with low power state (unaltered trace)

	Original	**Baseline**	**power-aware**
Total energy consumption (kWh)	23863	9467	8902
Renewable energy (kWh)	19329	7976	7529
Renewable energy (%)	81	84	85
Mean active CPU utilization	0.22	0.74	0.75
Mean active machines	100	29	27
Max active machines	100	43	42
Total tasks deferred	-	0	524
Mean task deferral time (minutes)	-	-	8.77
Max task deferral time (minutes)	-	-	35

Table 4.2: Overview of results with low power state (custom trace)

	Baseline	power-aware	Prediction algorithm (real results)
Total energy consumption (kWh)	10413	10001	9738
Renewable energy (kWh)	8662	8396	8199
Renewable energy (%)	83	84	84
Mean active CPU utilization	0.72	0.70	0.75
Mean active machines	33	32	30
Max active machines	60	59	62
Total tasks deferred	0	4601	23230
Mean task deferral time (minutes)	-	12.5	76
Max task deferral time (minutes)	-	60	393

4.6 Task prediction results

4.6.1 Prediction for low latency sensitive tasks

Prediction of mean CPU usage

The low latency sensitive network is trained with a batch size of 1000 samples, and iterates over the entire training set 200 times. In the training, a sample of the custom trace was used as validation data. The training loss on the training and validation data can be seen from figure 4.21(left). We see that the training loss on the validation set stabilizes after about 175 epochs, with an average loss of a little over 0.0045 for the last iterations.

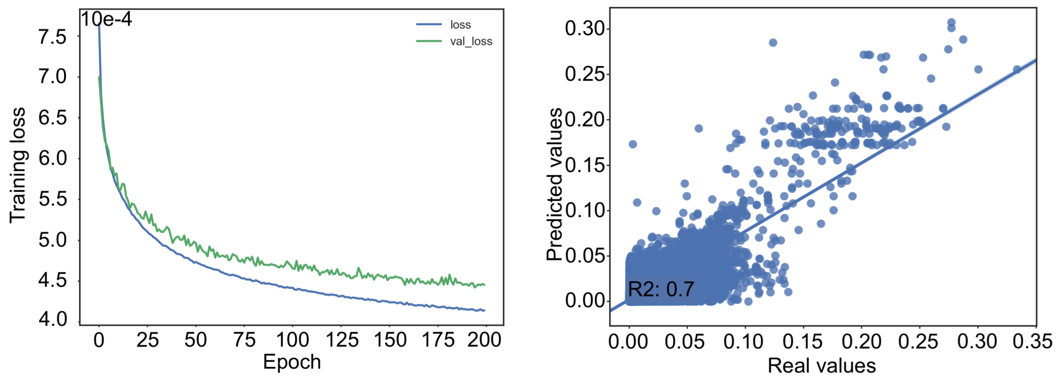

Figure 4.21: The learning curve (left) and the results of the prediction (right). In the learning curve, the blue line is the loss on the training data, and the green line is the loss on the validation set.

The mean prediction error of the trained network was on the test set was 0.0048

+/- 0.0076, and the median error 0.0022.

Compared to guessing at the mean value of the whole sample, this gives a much lower prediction error. Guessing at the sample mean would give a mean prediction error of 0.01.

The R^2 value of the fitted line from predicted and real values is 0.67, this is a satisfactory fit, although as seen in figure 4.21(right), there is quite a variance of error. This fit is the worst of all the neural networks, which is explained by this being the group with the most variance. This is also reflected in the real values, as the group of tasks with the lowest scheduling class also has a large variance of CPU usage.

Prediction of total duration

The network used for prediction of duration was trained with a batch size of 500 and iterated over the training set 300 times. Figure 4.22 (left) shows the training loss on the training set (blue) and on the validation set (green) during training. We see that the training loss is decreasing for most of the iterations but stabilize around 6 at 250 epochs. Figure 4.22 (right) shows the scatter plot of the real and the predicted values.

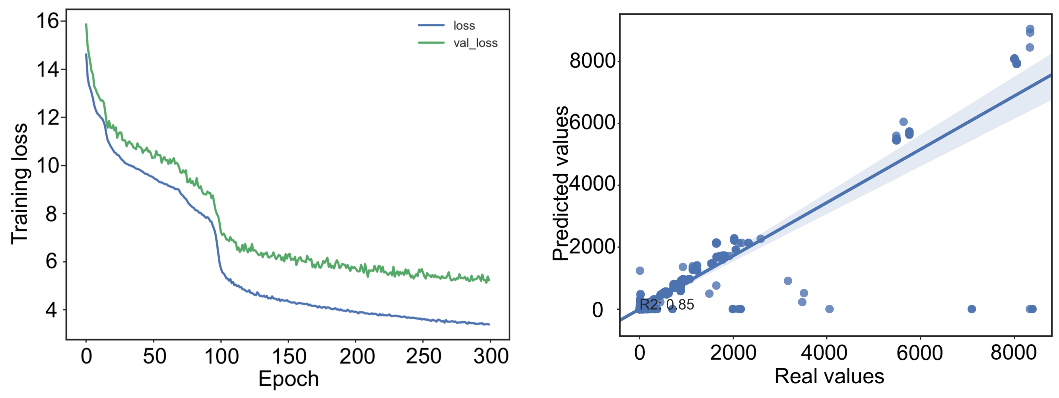

Figure 4.22: The learning curve (left) and the results of the prediction (right). In the learning curve, the blue line is the loss on the training data, and the green line is the loss on the validation set.

The mean prediction error on the test set was 5.34 +/- 65.30 (5 minute intervals), with a median error of 1 interval. The R^2 value of the fitted line from predicted and real values is 0.85. A median prediction error of 1 period is quite good, but as we can see from the plot of real and predicted values, most of the outliers are 0 when they should be higher.

The mean error of guessing at the mean of the samples are 24.2 intervals.

4.6.2 Prediction for medium latency sensitive tasks

Prediction of CPU usage

In this case, the fit results improved from the lower scheduling class, as can be seen from figure 4.23. Figure 4.23 (left) shows the learning error. Figure 4.23 (right) shows a scatter plot of the real and predicted values of mean CPU usage. The R^2 value of the line fitted to the points is 0.9. The best possible value of R^2 would be 1, so a fitted line with 0.9 is an acceptable fit, and it tells us that the points are centered around the regression line. The points have some variance, but the variance is centered. We can observe a steep decrease in the first epochs, which then flattens out around 0.003. This network was, as the low latency sensitive network, trained for 200 epochs with a batch size of 1000.

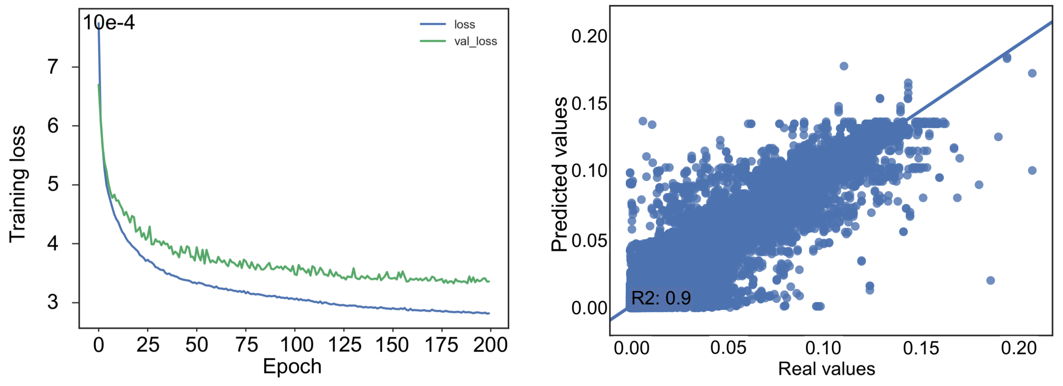

Figure 4.23: The learning curve (left) and the results of the prediction (right). In the learning curve, the blue line is the loss on the training data, and the green line is the loss on the validation set.

The mean prediction error of the network is 0.0034 +/- 0.0067, and the median error 0.0010. The mean error by guessing at the sample mean would be 0.016. Both the mean and median error is lower than guessing at the sample mean by a factor of 10.

Prediction of total duration

The best results for the medium latency sensitive duration prediction were obtained by having a batch size of 500 and this network was also trained by iterating over the training set 300 times. The learning curve in figure 4.24 (left) shows the loss on the training and validation data for each training iteration. The validation loss is higher than the training loss, which is as expected. After 200 iterations (or epochs), the validation loss is varying around 6, and seem to have stopped decreasing.

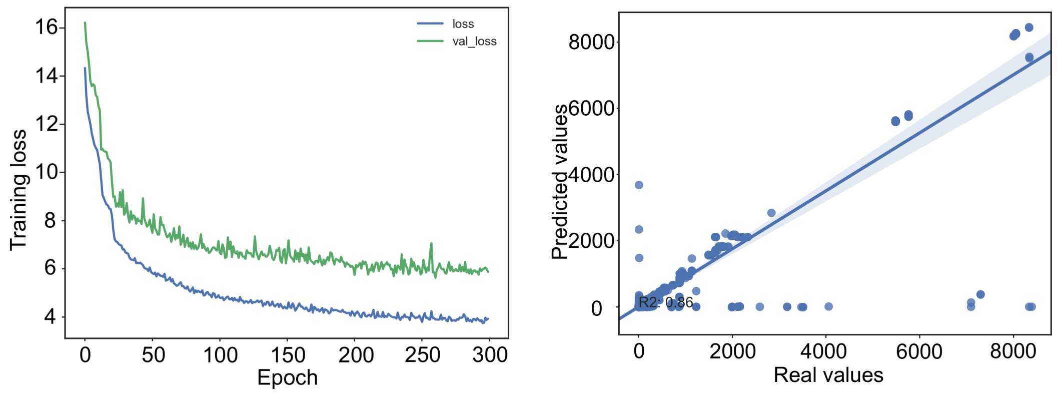

Figure 4.24: The learning curve (left) and the results of the prediction (right). In the learning curve, the blue line is the loss on the training data, and the green line is the loss on the validation set.

The mean prediction error for the network on the test sample was 6.05 +/- 72.14, with a median error of 1. In the prediction results in figure 4.24 (right), the R^2 score of the line fitted to the real and the predicted scatter plot is 0.86. We see that there are some outliers, and many of them are centered around 0.

If the predictions were made by guessing at the mean of the sample, the mean error would be 26.56 intervals, which is about 4 times worse than the prediction mean error.

4.6.3 Prediction for high latency sensitive tasks

Prediction of CPU usage

The prediction results, in this case, give an R^2 value of 0.89, as seen in figure 4.25 (right). The best possible value of R^2 would be 1, so a fitted line with 0.89 is a good fit. Figure 4.25 (left) shows the learning error. We see a steep decrease in the first epochs, and then it flattens out around 0.002. This network was trained with a batch size of 1000 and iterated over the training data 80 times.

The mean prediction error was 0.0021 +/- 0.0063, with a median error of 0.0005. The R^2 score of the regression line for the predicted data was 0.89, indicating

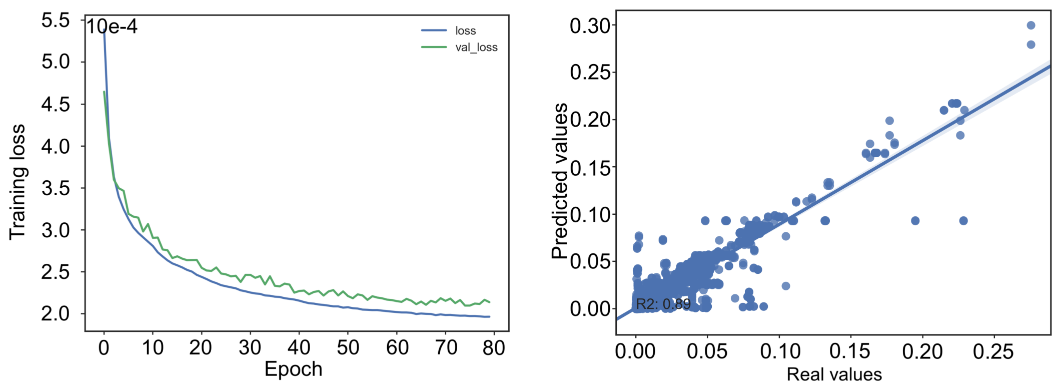

Figure 4.25: The learning curve (left) and the results of the prediction (right). In the learning curve, the blue line is the loss on the training data, and the green line is the loss on the validation set.

a good fit. This can also be seen from the lower spread of points far from the regression line.

Making predictions by guessing at the mean of the sample would yield a mean error of 0.012.

Prediction of total duration

The prediction of high latency sensitive duration was done with a batch size of 500 and the training data was iterated over 300 times during training. From figure 4.26 (left), we observe that the learning curve has a steep decrease in the first epochs, and then converging towards a final validation loss of 19.49 for the rest of the iterations.

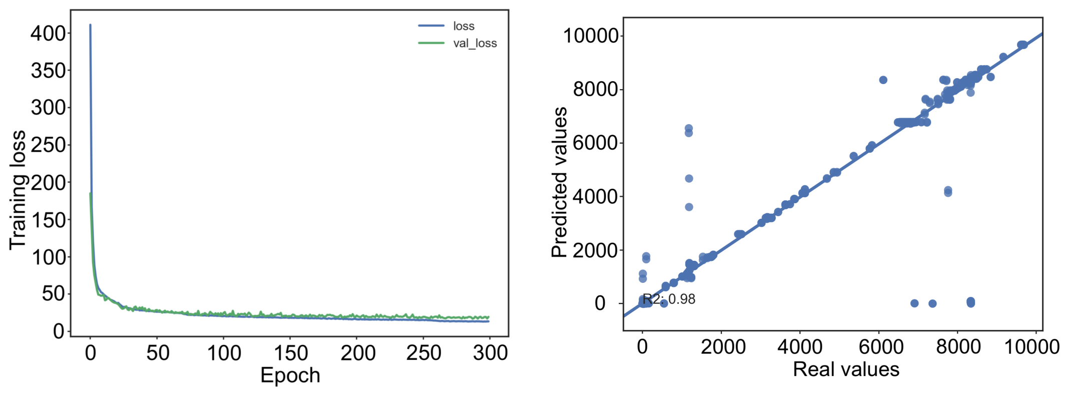

Figure 4.26: The learning curve (left) and the results of the prediction (right). In the learning curve, the blue line is the loss on the training data, and the green line is the loss on the validation set.

The mean prediction error of the network was 19.49+/- 274.98, with a median error of 1 interval. In figure 4.26 (right), a scatter plot of the real and predicted values for the test data is presented. The R^2 score of the fitted line was 0.98, corresponding to a very good fit. Most of the points in the scatter plot are centered around the regression line, and there are fewer outliers than for the former models.

Guessing at the mean would yield a mean error of 1248 intervals, much higher than the mean of the prediction error.

Chapter 5

Conclusions and further work

This chapter will summarize the content of this thesis, present the main findings and their impact, and will propose interesting avenues for further work on the subject.

5.1 Conclusion

In this thesis we have looked at how data centers can benefit from being co-located with a renewable energy source, and seen demonstrations on how a small scale data center co-located with a wind power source can adapt its workload to the variable power supply, and reduce its overall power consumption, to reduce its energy cost and environmental footprint.

Previous research has mainly focused on HPC workloads, often combined with solar power sources, and no individual task usage prediction. We have seen different approaches for both workload and power prediction, from none at all, to detailed prediction methods for the overall workload. This thesis contributes by looking at a combined workload with variable demand, using a neural network for individual task usage prediction, to optimally place tasks on the fewest machines possible in order to reduce energy consumption.

We presented a prediction algorithm that improves the data centers efficiency by using workload prediction on an individual task basis. This was done to avoid overloading machine capacity when placing tasks, thereby being able to run the machines at a higher capacity more of the time. This lead to a reduction of the total number of machines running at any time and to an overall reduction in energy consumption.

The algorithm also reduced the number of machines doing work in periods of low wind power production up to 79% of the time, by postponing tasks with a low latency sensitivity to a later time. It also reduced the overall power consumption from 23863 kWh for the original trace load to 9738 kWh with the prediction algorithm, by setting fully idle machines to a low-power state.

The prediction algorithm was compared to baseline results from a fully clairvoyant algorithm with respect to task usage and duration, both with and without taking power production into account. It was also compared to the original trace load from 100 machines.

The neural network used for prediction of the duration of the mean of the CPU usage of a task, and the duration of a task, shows good results. The predictions on higher scheduling classes show better results than the other groups, with an R^2 value of 0.89 and 0.98 for CPU usage and duration respectively. The medium latency sensitive tasks have R^2-values of 0.90 for CPU usage and 0.86 for duration, but the residual plot shows a large variation. The lowest latency sensitive tasks have R^2-values of 0.67 for CPU usage and 0.85 for duration.

The prediction method may not be suited for this problem in the real world scenario, deduced by the resulting oversubscription on machines for periods of time. The results indicate that the prediction algorithm reduces overall energy consumption, and increases the fraction of renewable energy used, compared to both the original trace load and to the baseline algorithms. The prediction algorithm has an 84% renewable energy consumption, while the original has 81% renewable consumption and the baseline has 83%. This increase comes at a cost of some machines running over their capacity for some time during the run periods.

The proposed algorithm also reduces the number of machines running in periods with low power availability up to 79% of the time.

Power calculations and task resource usage are based simply on mean CPU usage over 5 minute periods. Tasks real CPU usage fluctuates much quicker than this. In reality, there are also many more factors impacting the power usage of a data center, and the memory usage and other constraints of a task would also influence the placement of tasks on machines.

Despite the assumptions and simplifications considered in this work compared to the real world scenario, the results obtained can be seen as a proof of concept.

We proved that data centers would be a good candidate for utilizing excess wind energy. We demonstrated that data centers could also reduce their energy consumption by loading individual servers more on average, and this could be done by predicting individual task usage over time before placement.

5.2 Further work

To further validate the results found in this thesis, it would be interesting to test the algorithms on newer workload traces from other data centers. In October 2017, Microsoft released a workload trace from Microsoft Azure [29]. The trace is from 2016 and is from a data center running with virtual machines. A comparison of results from this thesis and the same methods applied to this newer workload would be an interesting next step, to establish if the methods still hold with a newer and more up-to-date workload trace.

It would also be beneficial to introduce more parameters into the algorithm for placing a task. Details on machine type, processing units, and other hardware details could make the algorithm closer to reality. It would also be interesting to include more parameters for power calculations, to better reflect the real-world scenario.

A better prediction algorithm, that takes CPU usage fluctuation into account, would be another interesting avenue for further work. The fraction of oversubscribed machines would likely be reduced with a more sophisticated prediction method for task CPU usage over time, where these fluctuations are better reflected in the prediction.

The number of machines running at times with low renewable energy availability could be reduced further by implementing a way to stop already running low latency sensitive tasks for these intervals, to make room for the higher latency-sensitive tasks, and put the stopped tasks back into the queue to be placed at a later time.

Since the results are based on a very small sample compared to the huge and complex structure of machines and connections a data center consists of, to verify the scalability of the method further testing with more parameters on a larger sample would be needed.

Bibliography

[1] Peter Corcoran and Anders Andrae. *Emerging trends in electricity consumption for consumer ICT (Technical Report)*. 2013.

[2] Gary Cook et al. *Clicking Clean: Who is winning the race to build a Green Internet? (Report)*. 2017, pp. 5, 15.

[3] Anders SG Andrae and Tomas Edler. "On global electricity usage of communication technology: trends to 2030." In: *Challenges* vol. 6, no. 1 (2015), pp. 117–157.

[4] *IEA statistics, World: Indicators for 2012*. URL: `http://www.iea.org/statistics/statisticssearch/report/?year=2012&country=WORLD&product=Indicators`.

[5] *Electricity domestic consumption*. URL: `https://yearbook.enerdata.net/electricity/electricity-domestic-consumption-data.html`.

[6] F. Yang and A. A. Chien. "ZCCloud: Exploring Wasted Green Power for High-Performance Computing." In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. May 2016, pp. 1051–1060.

[7] Íñigo Goiri et al. "Parasol and greenswitch: Managing datacenters powered by renewable energy." In: *ACM SIGARCH Computer Architecture News (Proceedings)*. Vol. 41. 1. ACM. 2013, pp. 51–64.

[8] Yanwei Zhang; Yefu Wang, and Xiaorui Wang. "GreenWare: Greening Cloud-scale Data Centers to Maximize the Use of Renewable Energy." In: *Proceedings of the 12th International Middleware Conference*. Middleware '11. International Federation for Information Processing, 2011, pp. 140–159.

[9] Luiz Andre Barroso; Jimmy Clidaras, and Urs Holzle. *The Datacenter as a Computer. An Inroduction to the Design of Warehouse-Scale Machines*. Second Edition. Morgan & Claypool Publishers, 2013, pp. 67–89.

[10] Matt Stansberry. *Uptime Institute's 2017 Data Center Industry Survey Results*. URL: `https://uptimeinstitute.com/webinars/2017_data-center_industry_survey_results`.

[11] David Meisner; Brian T Gold, and Thomas F Wenisch. “PowerNap: eliminating server idle power.” In: *ACM Sigplan Notices (Proceedings)*. Vol. 44. 3. ACM. 2009, pp. 205–216.

[12] Zhenhua Liu et al. “Renewable and Cooling Aware Workload Management for Sustainable Data Centers.” In: *SIGMETRICS Perform. Eval. Rev.* Vol. 40, no. 1 (June 2012), pp. 175–186.

[13] Annette Evans; Vladimir Strezov, and Tim J Evans. “Assessment of sustainability indicators for renewable energy technologies.” In: *Renewable and Sustainable Energy Reviews* vol. 13, no. 5 (2009), pp. 1082–1088.

[14] *Solar-PV power generation data*. URL: `http://www.elia.be/en/grid-data/power-generation/solar-power-generation-data/graph`.

[15] Nathan S. Lewis. “Toward Cost-Effective Solar Energy Use.” In: *Science* vol. 315, no. 5813 (2007), pp. 798–801.

[16] Andrew Krioukov et al. *Design and Evaluation of an Energy Agile Computing Cluster (Technical Report)*. UCB/EECS-2012-13. EECS Department, University of California, Berkeley, Jan. 2012. URL: `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-13.html`.

[17] C. Li; A. Qouneh, and T. Li. “iSwitch: Coordinating and optimizing renewable energy powered server clusters.” In: *2012 39th Annual International Symposium on Computer Architecture (ISCA) (Proceedings)*. June 2012, pp. 512–523.

[18] Md E. Haque et al. “GreenPar: Scheduling Parallel High Performance Applications in Green Datacenters.” In: *Proceedings of the 29th ACM on International Conference on Supercomputing*. ICS ’15. ACM, 2015, pp. 217–227.

[19] Baris Aksanli et al. “Utilizing Green Energy Prediction to Schedule Mixed Batch and Service Jobs in Data Centers.” In: *ACM SIGOPS Operating Systems Review* vol. 45.3 (2012), pp. 53–57.

[20] Jaeyeon Jung; Balachander Krishnamurthy, and Michael Rabinovich. “Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites.” In: *Proceedings of the 11th international conference on World Wide Web*. ACM. 2002, pp. 293–304.

[21] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks.” In: *Neural networks* vol. 4, no. 2 (1991), pp. 251–257.

[22] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” In: *CoRR* vol. abs/1412.6980 (2014). arXiv: `1412.6980`.

[23] *Google cluster data*. URL: `https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md`.

[24] *Google cluster-usage traces: format+schema*. URL: https://drive.google.com/file/d/0B5g07T_gRDg9Z0lsSTEtTWtpOW8/view.

[25] Daniel Gmach et al. "Capacity planning and power management to exploit sustainable energy." In: *Proceedings of the 2010 International Conference on Network and Service Management (CNSM)*. IEEE. 2010, pp. 96–103.

[26] Jonathan G. Koomey. *Estimating total power consumption by servers in the US and the world (Report)*. 2007. URL: https://www.greenbiz.com/research/report/2007/09/12/estimating-total-power-consumption-servers-us-and-world.

[27] *Efficiency: How we do it*. URL: https://www.google.com/about/datacenters/efficiency/internal/#tab0=25.

[28] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013, p. 74.

[29] Microsoft open source. *Microsoft Azure VM trace*. URL: https://github.com/Azure/AzurePublicDataset.