UNIVERSITÄT
MANNHEIM

Simulating a Data Centre's Flexibility in Power Demand

Masterarbeit
von

Nils Björn Wilken
Matrikelnummer 1404724

July 31, 2018

Vorgelegt am Lehrstuhl für Wirtschaftsinformatik II
Universität Mannheim
Gutachter: Prof. Dr. Christian Becker
Betreuer: Sonja Klingert

Eidesstattliche Erklärung:
Hiermit versichere ich, dass diese Arbeit von mir persönlich verfasst wurde und dass ich keinerlei fremde Hilfe in Anspruch genommen habe. Ebenso versichere ich, dass diese Arbeit oder Teile daraus weder von mir selbst noch von anderen als Leistungsnachweise andernorts eingereicht wurden. Wörtliche oder sinngemäße Übernahmen aus anderen Schriften und Veröffentlichungen in gedruckter oder elektronischer Form sind gekennzeichnet. Sämtliche Sekundärliteratur und sonstige Quellen sind nachgewiesen und in der Bibliographie aufgeführt. Das Gleiche gilt für graphische Darstellungen und Bilder sowie für alle Internet-Quellen. Ich bin ferner damit einverstanden, dass meine Arbeit zum Zwecke eines Plagiatsabgleichs in elektronischer Form anonymisiert versendet und gespeichert werden kann. Mir ist bekannt, dass von der Korrektur der Arbeit abgesehen werden kann, wenn diese Erklärung nicht erteilt wird

English version (not legally binding):
I hereby declare that the paper presented is my own work and that I have not called upon the help of a third party. In addition, I affirm that neither I nor anybody else has submitted this paper or parts of it to obtain credits elsewhere before. I have clearly marked and acknowledged all quotations or references that have been taken from the works of others. All secondary literature and other sources are marked and listed in the bibliography. The same applies to all charts, diagrams and illustrations as well as to all Internet resources. Moreover, I consent to my paper being electronically stored and sent anonymously in order to be checked for plagiarism. I am aware that if the declaration is not made, the paper may not be graded.

Mannheim, July 31, 2018　　　　Nils Björn Wilken

Contents

List of Figures

List of Tables

List of Abbreviations

APC	Average Power Consumption
CPP	Critical peak pricing
CPU	Central Processing Unit
DC	Data Centre
DOE	Department of Energy
DR	Demand Response
DVFS	Dynamic Voltage and Frequency Scaling
EP	Energy Provider
EtS	Energy to Solution
EU	European Union
ESF	Energy Storage Facility
GHz	Gigahertz
HDD	Hard Disk Drive
HPC	High Performance Computing
HVAC	Heating, Ventilation, and Air Conditioning
kW	Kilowatt, Kilowatts
kWh	Kilowatt-hour, Kilowatt-hours
LBNL	Lawrence Berkeley National Laboratory
LRZ	Leibniz Supercomputing Centre
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MW	Megawatt, Megawatts
PowerDAM	Power Data Aggregation Monitor
PDU	Power Distribution Unit
PEC	Energy Provision Compensation
PECP	Provided Energy Compensation Price

PPCP	Provided Power Compensation Price
PPC	Power Provision Compensation
PUE	Power Usage Effectiveness
RAM	Random Access Memory
RTP	Real-time pricing
SLA	Service Level Agreement
STDF	Shortest Time To Deadline First
TDP	Thermal Design Power
TOU	Time of use pricing
UPS	Uninterruptible Power Supply
W	Watt, Watts
WSC	Warehouse Scale Computers

1. Introduction

In times of the digital revolution, the required amount of computing power has increased heavily over the past and will steadily increase in the future. In 2020 the number of installed cloud workloads worldwide will approximately increase to 478.8 million [Net16]. This embodies an increase of approximately 265%, compared to the 180.9 million workloads that were installed back in 2015 [Net16]. To service these growing workload numbers, data centers have to grow bigger and bigger in terms of hosted servers. This development causes several major problems. With the size of a data center, also the energy consumption of the data center increases. In 2015, the data centers of the world already consumed about 416.2 terawatt hours of electricity, which embodies approximately 3% of the world's energy production [ind]. This amount is likely to grow to even larger dimensions in the near future.

This is a major problem, because a major fraction of the total energy generation uses fossil or nuclear energy sources. In 2017 fossil energy sources were used for 58.7% of the worldwide electricity generation [Age18], where renewable sources only provided a fraction of 9.7% [Age18]. As electricity generation from fossil and nuclear sources has a negative impact on the environment and especially fossil sources speed up the climate change on earth, the energy sector currently tries to increase the fraction of renewable sources for electricity generation.

However, this transformation raises new challenges for the electricity generators and grid operators. In contrast to power plants fueled by fossil or nuclear sources, the resources used by the renewable generation plants cannot be artificially controlled, because they often rely on natural resources (wind, sunlight) which are not controllable. As a consequence, the amount of electricity that is generated by renewable power plants depends on the current availability of the renewable resource. This introduces a considerable amount of power supply volatility into the power grid. As the power grid could fail when more power is supplied than demanded or more power is demanded than supplied, the grid operators have to compensate for the power peaks and power lows in the grid. This induces some regulation costs for the grid operators. To compensate power lows, the grid operators commonly use very inefficient but highly reactive diesel generators. In the case of power peaks, the unused energy is wasted through the operation of some additional electricity consumers. The need for this kind of compensation actions, reduces the positive impact of the use of renewable energy sources for electricity generation. An innovative approach to reduce the need for such compensation actions is called Demand Response (DR). Grid operators that use this approach try to incentivize large energy consumers, like for example large scale data centres (DC), to adjust their power consumption profile, in order to compensate for a part of the volatility that is introduced by the renewable energy sources. When enough energy consumers would participate in this approach, the need for compensation actions, which keep the grid in a stable state, would be mitigated completely.

1.1. Problem Definition

Recently, due to the steadily growing power demand and the high degree of controllability of large DCs, DCs are considered to be very well suited candidates for a participation within DR programs [WLLMR14]. Although DCs provide many characteristics (cf. Section 2.1.4) that are beneficial for a participation within DR programs, one characteristic might be a prohibitive factor. The customers of large commercial DCs usually pay some usage fee for the usage of the services, it is in great interest of the customers that the services are provided on or above some reasonable service level. Therefore, most large commercial DCs negotiate a service level agreement (SLA) with their customers, which specifies the minimum service level. In case of a SLA violation, the DC is obliged to pay some financial compensation to their customers [GTGB14]. SLAs might be a prohibitive factor for DR program participation, because most techniques for power consumption reduction imply service level degradations and thus, might imply compensation payments to the DC's customers. When the incentives earned for the provision of power demand flexibility are smaller than the possible compensation fees, this leads to an operative loss for the DC. As a consequence, rational DC operators will have no incentive to participate in a DR program.

1.2. Objective and Approach

The main objective of this thesis is to develop a simulation framework that is able to realistically simulate the impact of the application of several power demand flexibility provision techniques onto the power consumption profile, the SLA costs, and the energy costs of a large scale DC. Another objective is to evaluate whether a large scale DC can increase its revenue by the participation within one or several DR program(s). A third objective of this thesis is to find an optimal combination of power demand flexibility provision strategies, that maximizes the revenues that a large scale DC can earn by the participation in an exemplary DR program.

Due to practical and time constraints of this thesis, the different combinations of strategies cannot be tested in a real world DC environment. Instead, during this thesis an adjusted version of DCSim, which is a java-based data center simulation framework that was originally implemented for the use in the EU funded All4Green[1] project, is developed. The adjusted version will be used to simulate the participation of a large scale DC within an exemplary DR program. For the simulation, this thesis utilizes a real world data trace from a scientific high performance computing (HPC) center in Germany.

1.3. Structure of the Thesis

Chapter 2 provides some theoretical foundations that are used throughout the thesis. Chapter 3 summarizes important related work. Chapter 4 describes the general simulation scenario and the available real world data traces. Chapter 5 analyzes the original status of the DCSim simulation framework. Chapter 6 explains the adjustments that were made to DCSim. Chapter 7 introduces the different models that are implemented in the adjusted version of DCSim. Chapter 8 validates the adjusted simulation framework and evaluates the potential of a large scale DC to participate in an exeplary real world DR market. Chapter 9 provides a conclusion of the thesis, highlights the limitations of this thesis, and provides some suggestions for future work.

[1]`http://all4green-project.gfi.es/`

2. Theoretical Foundations

This chapter establishes a common theoretical foundation for the discussion in the rest of this thesis. Therefore, Section 2.1 introduces the concept of DR. Subsequently, Section 2.2 provides an introduction into the data centre (DC) domain.

2.1. Demand Response

This section highlights some important aspects of the DR concept.

2.1.1. Demand Response in the United States

According to the Department of Energy (DOE) of the US, DR "is a tariff or program established to motivate changes in electric use by end-use customers in response to changes in the price of electricity over time, or to give incentive payments designed to induce lower electricity use at times of high market prices or when grid reliability is jeopardized" [DoE06]. Therefore, DR is able to increase the grid stability by adjusting the overall power demand to variable levels of available power supply. It also reduces the local network investment costs, because through the adjustment process, the total peak power that has to be transmitted is reduced [BZBK+16].

In 2005, the US enacted an Energy Policy Act which defines that it is the goal of the US to foster the development of DR withing the different states of the US [Gov07].

2.1.2. Demand Response in Europe

According to Bertoldi *et al.* (European Commission), DR "is a tariff or programme established to incentivise changes in electric consumption patterns by end-use consumers in response to changes in the price of electricity over time, or to incentivise payments designed to induce lower electricity use at times of high market prices or when grid reliability is jeopardised" [BZBK$^+$16].

In 2012, the European Union (EU) defined first official guidelines that aim at the development of DR in all member states of the EU. This is done in the Energy Efficiency Directive (2012/27/EU) [tC12]. Thus, DR in Europe is still in its infancy and started much later than in the US.

2.1.3. Types of Demand Response Programs

DR programs can be subdivided into three major categories [Sia14]:

Rate or price-based programs: In DR programs of this type, power consumers are incentivized to adjust their power consumption via a change of the electricity price. As power consumers are likely to consume more power when the electricity price is rather low, the utilities will adjust the price such that it is high during peak intervals and low during off-peak intervals [Sia14]. The temporal details (e.g., whether prices are determined in real time or in a day-ahead manner) may change from program to program [Sia14].

Possible price options are TOU (time of use rates), CPP (critical peak pricing), and RTP(real-time pricing) [Sia14]. In TOU, the electricity prices are specified for fixed intervals that are spread across the day. When a program uses CPP, the grid operators define several hours per day as critical peak time. Within this critical peak time intervals, the grid operators will charge the consumers a predetermined peak price, which is usually very high. In programs that use RTP, the price changes (usually hourly) dependent on the current supply and demand on the market [Sia14].

Incentive or event-based programs: In incentive-based DR programs, the power consumers receive some reward from the grid operators, when they adjust their power consumption in response to a DR event, which is issued by the grid operators. The reaction to a DR event request can be either mandatory or voluntary, depending on the specific DR program. In DR programs of this type, there usually exist upper limits on the length of the power adjustment intervals and the total amount of event hour per year [Sia14].

Some possible examples for incentive based programs are emergency DR programs, capacity market programs, and ancillary services market programs [Sia14]. In emergency DR programs, the participants receive a financial reward for reducing their load in order to increase the grid stability. Very similar to this, in capacity market programs, the participants receive a financial reward for reducing their load in times in which more grid capacity is needed. In ancillary services market programs, the participants receive a financial reward for responding to load adjustment requests, which are issued by the grid operators. This supports the grid operators to properly operate the grid [Sia14].

Reduction bid based programs: In reduction bid based programs, the power consumers determine how much power flexibility they are willing to offer in a specific time interval and the reward for which they would offer this flexibility. These two values are then aggregated in a bid, which is transmitted to the grid operator [Sia14].

2.1.4. Demand Response in Data Centres

As already mentioned in Section 1.1, DCs are considered to be very well suited for the provision of power demand flexibility [WLLMR14]. There are mainly two reasons for this. First, modern DCs consume very large amounts of electricity (very large DCs can consume up to 50MW [WLLMR14]) and secondly, DCs are very well monitored and highly automated, which makes DCs to very flexible power consumers [WLLMR14]. Some evidence for the power demand flexibility of DCs is given by the Lawrence Berkeley National Laboratory (LBNL). In some empirical studies, they showed that four exemplary DCs are usually able to cut 5% of their load within 5 minutes and 10% of their load within 15 minutes [GPF$^+$09], [GGMP12]. These numbers were achieved without touching the IT workload of the DCs. Thus, if one would also incorporate changes of the IT workload of the DCs, the flexibility would even increase [WLLMR14]. Nevertheless, in practice not many DCs participate actively within DR programs [GGMP12]. There might be several reasons for this, however, one of the major reasons might be that the existing DR programs do not fit the requirements of large commercial DCs [WLLMR14].

In the past years, a lot of research effort has been invested into the development of technical approaches, which can be used to exploit the power demand flexibility of DCs [WLLMR14]. The relevant ones for this thesis are Dynamic Voltage and Frequency Scaling (DVFS) and Workload Shifting.. The ideas of these approaches are as follows:

Dynamic Voltage and Frequency Scaling (DVFS): DVFS is a very common and effective technique that can be used to reduce the power consumption of a CPU. The main idea of DVFS is to alter the voltage supply level of the CPU, which will reduce or increase the frequency of the affected CPU. Through the control of the voltage supply level, the power consumption of the CPU is controlled as well. Nevertheless, when the frequency of a CPU is decreased, the runtime of the executed workload will increase. This might lead to possible SLA violations [CSP05].

A schema of the concept of DVFS is illustrated in Figure 2.1. The gray boxes represent the power consumption profile of an exemplary job. The red line represents a deadline before which the job should be finished to avoid possible SLA violation costs. The x-axis of the two schemata represents the time axis and the y-axis represents the power axis. Both, the schema on the left hand-side and the schema on the right hand-side of Figure 2.1 can be chosen as the starting scenario. In each case, the power consumption behavior of the job, which is illustrated on the corresponding opposite side, can be reached via the adjustment of the CPU frequency in the direction that is indicated by the arrows in the center of Figure 2.1.

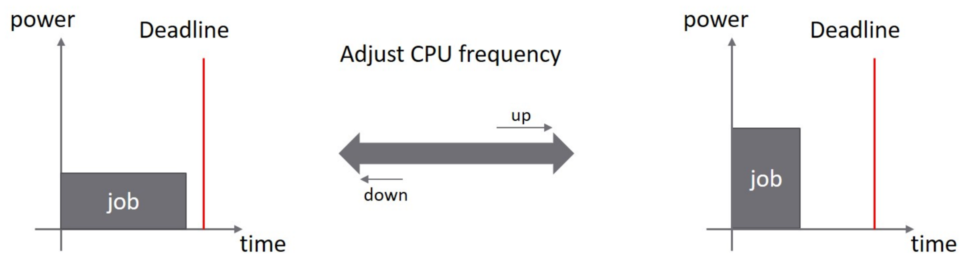


Figure 2.1.: DVFS concept.

Workload shifting: Usually, DCs are considered to execute a mixture of interactive and batch workloads [QKP+09]. The interactive workload is not delay tolerant, which means that it cannot be shifted in time. In contrast, batch workload is considered to be delay tolerant and thus, can be shifted in time [GTGB14]. By shifting delay tolerant workload in time, the power that is required to execute the shifted workload is shifted as well and the power consumption at the timestep, where the workload was originally scheduled, will decrease.

A schema of the concept of workload shifting is shown in Figure 2.2. The axes, gray boxes, and red lines have similar meanings as the corresponding components in Figure 2.1. In addition, the plots in Figure 2.2 contain a green line which indicates the overall power consumption profile of all jobs. Each of the two illustrated schemata can be picked as the starting scenario. In both cases, the total power consumption profile of the corresponding plot on the opposite side of the figure can be achieved by the utilization of the actions which are indicated by the two arrows in the center of the figure.

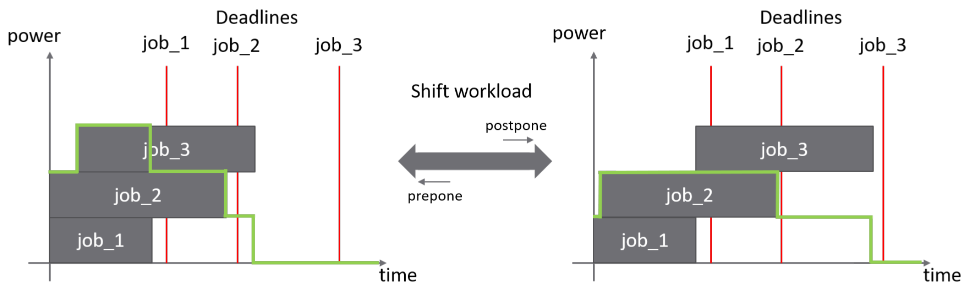


Figure 2.2.: Workload shifting concept.

2.2. Data Centres

This section introduces the theoretical basics of the DC environment.

2.2.1. Data Centre Characteristics

According to Barroso *et al.* “A datacenter is a building (or buildings) designed to house computing and storage infrastructure in a variety of networked formats” [BCH13, p.47]. The main functions of a DC are to fulfill the needs of the housed equipment and personnel which are reliable provision of power, sufficient cooling, shelter, and security . The operational core functionality of the DC building is to provide the installed hardware with energy and to remove the resulting heat from the building. The execution of this process drives the largest fraction of the DC's operational costs. Thus, a DC's operational costs are usually proportional to the total amount of consumed energy [BCH13].

In 2013 there were approximately 51100 DCs located in Germany. About 95% of these DCs are smaller than 100 square meters. However, it is estimated that this will change and that the amount of DCs which are larger than 100 square meters will increase significantly in the future [HC14]. In 2012, the energy consumption of all DCs in Germany was around 9.4 TWh, which is approximately 1.8% of the total energy consumption of Germany [HF13].

A possibility to distinguish DCs is the kind of workload they process. So called Warehouse Scale Computers (WSCs), which are widely used in the commercial cloud computing sector (Amazon EC2, Microsoft Azure, ...), execute mostly internet services that aim to have an uptime of 99.99%. In contrast, High Performance Computing (HPC) DCs, are mainly used to execute scientific workload like large weather simulations. The scientific applications, which are executed by HPC DCs, are usually much less diverse and much more synchronized than internet services. Thus, the scientific workloads produce much more communication overhead to achieve the high level of synchronization. Another important difference is the runtime of the workload. Scientific applications can run for hours or even several days, which leads to a constant utilization of the DC during this period. In contrast, the utilization of a DC, which executes internet services, often follows a diurnal pattern and very rarely crosses the 90% mark [BCH13]. The DC that is in the focus of this thesis is a typical HPC DC which executes only scientific workloads.

2.2.2. Energy Consumption of a Typical Data Centre

A DC houses several components that consume electrical power. The most important ones and a simplified power distribution hierarchy are illustrated in Figure 2.3. The main power supply from the utility and the power supply of the backup generator both feed into a switch gear device, which forwards the currently active supply to the DC's infrastructure. The backup generator is only active, when there is an outage of the utility power supply. The lights, office space, and cooling equipment are directly supplied by the switch gear. In contrast, the IT equipment of the DC uses so called uninterruptible power supplies (UPSs) and power distribution units (PDUs) in between. An UPS is a device that contains an additional battery that is able to replace the main power supply in case of a power outage until the backup generator is fired up. A PDU distributes the power, which is supplied to it, to several different IT equipment racks [BMN+07].

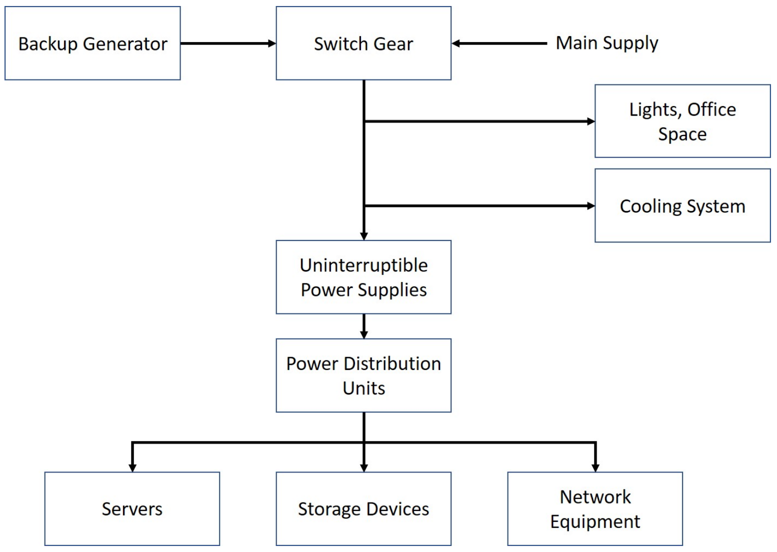


Figure 2.3.: Simplified Power Distribution Hierarchy of a DC, inspired by [BMN+07].

Figure 2.4 shows a breakdown of the energy consumption of the different components within a DC. One can see that the cooling equipment consumes about 50% of the total energy consumption of the entire DC. The second largest fraction is consumed by the IT equipment (network, server, and storage equipment). It is also interesting to note that a not insignificant fraction of 11% of the total energy consumption is lost through conversion and distribution losses. The lighting of a DC consumes only a small fraction of 3% of the total power consumption [DWF16].

$$Energy = AveragePower * Time \Leftrightarrow AveragePower = \frac{Energy}{Time} \tag{2.1}$$

It is important to note that power and energy are different units of measurement. The power unit quantifies the amount of electricity that an electricity consumer

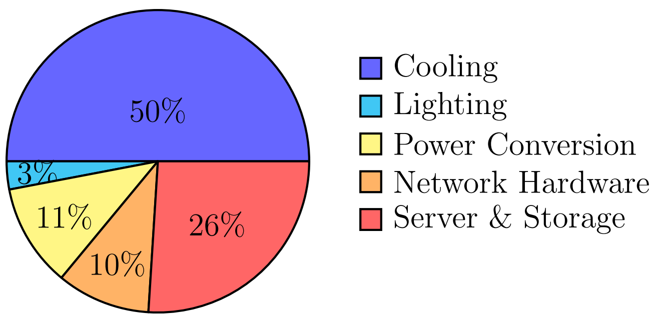


Figure 2.4.: Breakdown of the energy consumption of different components within a DC [DWF16].

requires to operate. The energy unit quantifies the average amount of power that was consumed over a period of time. Equation 2.1 illustrates the connection between the two units.

In the case of single DC hardware components (like a CPU), the power consumption is usually specified in watts (W). Energy can be quantified in any required time resolution (seconds, minutes, hours, etc.). Usually, the utilities charge their customers based on kilowatt hour (kWh) prices.

One important association in the context of the development of energy efficient DCs is The Green Grid. The Green Grid is an association of IT professionals that aim to increase the energy efficiency of DCs dramatically through the publication of several proposals [BRPC08]. In 2008, The Green Grid defined a measure that is supposed to quantify the energy efficiency of a DC. This measure is called Power Usage Effectiveness (PUE) and is defined as follows [BRPC08]:

$$PUE = \frac{TotalFacilityPower}{ITEquipmentPower} \tag{2.2}$$

Where $TotalFacilityPower$ is the power measured at the electricity meter of the utility and the $ITEquipmentPower$ is the power that is used to operate the hardware components of a DC that are needed to process data. Theoretically, the PUE ranges from 1.0 to infinity. Thereby, a PUE of 1.0 is considered to be optimal, because in that case the entire power consumption of the DC would be used to operate the IT equipment and no power is “wasted” for cooling or other overhead processes [BRPC08].

3. Related Work

This thesis is located at the meeting point of two fields of related research. These fields are namely *Demand Response in Data Centres* and *Data Centre Simulation*. This chapter provides a short introduction to both of these fields of research and highlights the contribution of this thesis.

The remainder of this chapter is structured into three sections. Section 3.1 introduces to the field of *Demand Response in Data Centres*. Section 3.2 gives an overview over the field of Data Centre Simulation. Finally, Section 3.3 highlights the main contribution of this thesis.

3.1. Demand Response in Data Centres

The research that addresses DR in DCs can be basically grouped by the different types of DR programs. Much of the research in this area focuses on rate or price-based DR programs and thus, try to minimize the energy cost of a DC. For example, Yao and Zhang [YLZ14] used on site batteries to buffer energy. The batteries were charged when electricity prices are low and discharged when electricity prices were high. Via a simulation they found that the application of their approach can save about 0.03$ per server in 24 hours. Aksanli and Rosing [AR14] also use energy buffering via batteries to reduce the energy costs of a 21MW DC. The authors found that the application of their peak shaving method can provide energy costs savings of 480000$ per year for the modeled test DC. Liu *et al.* [LWC$^+$13] try to minimize the electricity costs by avoiding the critical peak intervals through the use of workload shifting and local renewable energy generation. They showed that the application of their technique can provide cost savings of up to 40% for their test site Fort Collins Utilities, Colorado.

There is also a lot of research that focuses on the incentive or event-based DR programs. For example, very recently, Cioara *et al.* [CAB$^+$18] conducted an experiment where they developed an approach that utilizes workload shifting, thermal storage facilities, and batteries within a DC to provide some power demand flexibility. Via a simulation the authors showed that DCs can adjust their power load profile according to the demands of the tested DR markets. Ghasemi-Gol *et al.* [GGWP14] developed an approach that uses load shedding, which is achieved by switching off/on servers to adjust the power consumption. They found that their test DC can reduce the energy costs by up to 13% when the approach is applied. Another interesting study is conducted by Tang *et al.* [TDC+14]. In their study they examine the potential for a very small DC (300kW) to participate within a DR program. For the power demand provision they consider to change the temperature setpoint of the cooling equipment and to select the optimal workload dispatch algorithm. They found that also very small DCs have large potential to offer ancillary services, but cannot do so due to the entry requirements of the DR markets.

3.2. Data Centre Simulation

In recent years the use of cloud services increases heavily and thus, the DCs, which host these services, tend to grow in size and amount of consumed energy. As the amount of consumed energy makes up a large portion of the total operational costs of a DC [RRT+08, FWB07], a lot of effort was invested into the development of energy- and workload management approaches to make DCs more energy efficient [KBK12]. To test these approaches within a real world DC environment is extremely expansive, as this would induce the same costs as the productive operation of the DC. In addition, it is very complex and time consuming to reconfigure a large scale real world DC environment [CRB+11]. It is much cheaper to use a cloud DC simulation environment instead [CRB+11]. Within these framework it is easily possible to test the developed approaches and to reconfigure the whole DC infrastructure for different test runs [CRB+11].

Ostermann *et al.* proposed a simulation framework which is called *GroudSim* [OPPF10]. *GroudSim* is a simulation framework for scientific workloads that relies on a discrete-event simulation core. The framework provides some basic analysis features for the evaluation of the simulation runs and allows the user to model computational and network hardware, job submissions, component failure, background load, and DC costs. Meisner *et al.* developed the *BigHouse* [MWW12] simulation framework. Instead of a micro-architectural model for each server, *BigHouse* uses a combination of several queuing theory and stochastic models to simulate the DC. Thereby, *BigHouse* uses a distributed discrete-event simulation core. *BigHouse* is able to record several performance metrics up to a prespecified statistical significance. Kliazovich *et al.* introduced the *GreenCloud* framework [KBK12], which is an extension to the Ns2 network simulator [ns2]. The focus of the simulator is to record the power consumption of DC components and thus, also to record the energy costs of a DC. *GreenCloud* can be used to simulate two-tier, three-tier, and three-tier high-speed DCs. Another simulation framwework, which is called *EMUSIM*, was developed by Calheiros *et al.* [CNRB13]. In contrast to the previously mentioned framworks, *EMUSIM* is a combination of a simulator and an emulator. The emulator of *EMUSIM* to create application profiles that are used by the simulator. The authors designed this framework to improve the evaluation of application behavior when executed on cloud DCs. The simulation part of *EMUSIM* is based on *CloudSim* [CRB$^+$11], which is another DC simulation framework that was developed by Calheiros *et al.*. *CloudSim* is also the basis of the simulator that is utilized for this thesis. *CloudSim* enables the user to model system resources as well as the behavior of each DC component (e.g., virtual machine and resource provisioning). The framework also provides the possibility to model inter-networked federations of several DCs and not only the simulation of a single DC. According to the authors, the framework is also used by large resource organizations in the US like HP Labs. Another extension to the *CloudSim* framwork is the *NetworkCloudSim* [GB11] framework, which was developed by Garg and Buyya.

3.3. Contribution of this Thesis

So far, much work has been done separately in the related fields of reasearch. However, thus far not much effort was invested into the possible combination of theoretical DR approaches for the DC domain with a powerful DC simulation framework. Such a framework would provide the possibility to validate the feasibility of newly developed DR approaches with very low effort. This thesis contributes to this issue by the development of a DC simulation framework that implements the necessary functionality to simulate the application of two power demand flexibility techniques within a DC.

4. Simulation Scenario

This chapter describes the simulation scenario and the realistic data traces which are used for the evaluation of this thesis. Furthermore, is summarizes the assumptions that are made for the simulation scenario in this thesis. Therefore, this chapter is structured into three sections. Section 4.1 briefly describes the DC that is considered for this thesis. Section 4.2 describes all data traces that were provided by the DC. Finally, Section 4.3 summarizes the assumptions that are made for the simulation scenario in this thesis.

4.1. Considered Data Centre

For the evaluation of this thesis, phase 1 of the SuperMUC, which is a large scale HPC system that is deployed at the Leibniz Supercomputing Centre (LRZ) in Germany, is considered. It is an IBM[1] iDataPlex DX360M4 Intel Sandy Bridge system with 3.18 PetaFLOPS theoretical peak performance [SWHB17]. According to the TOP500 list [TOPa], in 2012 the SuperMUC was the fastest supercomputer in Europe and the fourth fastest in the world. It has an efficiency of 0.85 GFLOPS/W [TOPb]. The SuperMUC phase 1 contains 9216 compute nodes that are distributed over 18 node islands. Each node contains two Intel Sandy Bridge-EP Xeon [Int] E5-2680 8C octa-core processors. Thus, the SuperMUC phase 1 has 147456 processor cores in total. Each of the processors has a maximum operating frequency of 2.7 GHz and a maximum turbo frequency between 3.1 GHz and 3.5 GHz. Furthermore, each processor has a thermal design power (TDP) of 130W [SWHB17]. The TDP quantifies a power value that an electric component should not exceed during operation. In 2014, the total energy consumption of the SuperMUC was roughly 20000MWh [SWLB17].

The SuperMUC mainly processes scientific applications, as it is part of a scientific organization. For the management of the resources and to schedule the

[1]`https://www.ibm.com/`

workload, the SuperMUC uses the LoadLeveler[2] system, which is developed by IBM. The policies of the SuperMUC define that a job is executed with a default frequency of 2.3GHz. Nevertheless, applications can also be executed on a higher frequency when they are sufficiently optimized. Applications that achieve a runtime degradation of 5% when executed with 2.5GHz instead of 2.3GHz, will be executed with 2.5GHz. When an application achieves a runtime degradation of 12% when executed with 2.7GHz instead of 2.3GHz, it will be executed with 2.7GHz [ABB+14].

4.2. Description and Preprocessing of the Data Traces

This section describes the data traces that were provided from the SuperMUC, which is described in the previous section, in the year 2014. All the data were obtained by the use of the Power Data Aggregation Monitor (PowerDAM) toolset [SWAB14].

4.2.1. Job Data

The Job Data data trace contains information for every job that was executed on the SuperMUC in the year 2014. In total it contains records for 406980 jobs. Each record consists out of 10 fields which contain the following information:

Job id: This field contains a distinct job id which the job was assigned by the scheduling system.

Submission time: This field contains the time at which the job was submitted to the scheduling system.

Start time: This field contains the time at which the actual execution of the job started.

End time: This field contains the time at which the actual execution of the job finished.

[2]`https://www.ibm.com/support/knowledgecenter/en/SSFJTW_5.1.0/com.ibm.cluster.loadl.v5r1.load100.doc/am2ug_ch1.htm`

Status: This field corresponds to the status of a job within the LoadLeveler system. Within the available data trace, for this field exist two different values which are *Completed* or *Removed*. The *Completed* value indicates that the corresponding job was executed successfully. In contrast, the *Removed* value indicates that the corresponding job was not executed successfully, but was removed from the system. However, in theory there are much more statuses that a job can take within LoadLeveler, a full list can be found at [Cen].

Energy tag: The value in this field is a user specified job tag. In theory this value should be unique for each application. However, as the values are freely specified by the users of the DC, who do not really care about the uniqueness of these tags, one can assume that these tags cannot be used to identify the executed application of a job.

CPU frequency: This field states the maximum CPU frequency that was allowed for the execution of the corresponding job. In practice, however, it depends to a large extend to the nature of the job, whether this maximum CPU frequency is close to the actual average frequency during execution or not. For example, memory bound applications are likely to run on a much lower frequency in practice, whereas CPU bound applications are likely to be executed on a frequency that is very close to the maximum allowed frequency.

Number of utilized nodes: The number in this field indicates the number of compute nodes which were utilized for the execution of the job. Here it is important to note that a compute node is the smallest unit that a job can request during the execution. Thus, it is not possible that the two CPUs of one node are assigned to two different jobs at the same time.

Energy to solution (EtS): This field states how much energy the corresponding job consumed during execution. The values are specified in kilowatt hours and directly measured by the LoadLeveler system during execution time.

Average power consumption (APC): In this field, the average power consumption of the corresponding job is quantified. The values are specified in watts and are not directly measured. Instead, they are calculated from the runtime and the EtS values of the jobs.

There are some job records within the data trace that have zero values for the frequency, the EtS and the APC values but positive runtime. This might be due to the abortion of those jobs or due to measuring mistakes. Nevertheless, as these records might influence the simulation results, they were deleted from the data trace. Furthermore, all records that had a negative runtime were deleted. In addition, as the idle power of the CPUs that are used in the nodes of the DC is approximately 23W [HTHW16] and each node contains two CPUs, the idle power of a compute node should at least be 46W. As this base power can slightly variate [HTHW16], all records that had an APC/Node value of less than 40W were removed. The vast majority of these records had a runtime of less than 5 seconds, which suggests that the jobs were aborted, which might lead to measuring mistakes that cause the low APC/Node values. After the data cleaning, the job data trace contained 389968 job records.

Table 4.1.: Job data trace statistics.

	EtS(kWh)	**APC(W)**	**Runtime(h)**	**Frequency(GHz)**	**#Nodes**
Maximum	18485.1	2071975.68	52	2.7	9216
Minimum	0	40.5	0.0002778	1.2	1
Median	0.02	274.55	0.104	2.3	2
Average	26.65	3815.04	3.6	2.39	31.2
Std. Dev.	239.92	26211.18	9.96	0.17	233.65
Skewness	31.49	31.97	3.41	-0.02	27.62

Table 4.1 displays statistics of several variables from the cleaned job data trace. The average runtime of a job was 3.5 hours, whereas the maximum runtime was 52 hours. So there exists a large variety of different application runtimes and thus, also applications. Also important to note is that the median runtime (0.104h) was much lower than the average runtime, which indicates that most of the jobs had a rather short runtime. The same holds also true for the EtS and APC values. The maximum values for EtS and APC were 18458.1kWh and 2071975.68W. The corresponding average values (26.08kWh and 3732.9W) and median values (0.02kWh and 274.55W) were much lower. That most of the jobs had below average values for EtS and APC is also indicated by the high skewness of both variables. As expected, because the DC's policy sets 2.3GHz as default frequency, the average frequency was 2.38GHz which is very close to the default frequency and the median value exactly matches the default frequency. Also not

surprising is that the maximum frequency was at 2.7GHz, as this is the maximum frequency of the CPUs without turbo mode. However, as the minimum observed frequency was 1.2GHz, some jobs were executed on lower frequencies than the default frequency. The maximum amount of nodes that a job utilized in 2014 was 9216, which is the entire available amount of nodes. In contrast, the average (31.2 nodes) and median (2 nodes) values were much lower, which indicates that the majority of the jobs only required a tiny fraction of the available nodes. The high skewness also indicates that the majority of the jobs utilize a number of nodes that is below the average.

4.2.2. PUE and IT Power Time Series

The two data traces for PUE and IT Power are time series data. They contain hourly values for the PUE respectively the IT Power consumption of the SuperMUC for the complete year of 2014.

The IT power consumption values are stated in kilowatt hours. The IT power consumption, which is illustrated in Figure 4.1, was measured at the main power lines that supply the section of the DC that contains servers, storage, network, internal cooling pumps, and PDUs. In 2014 the average IT power consumption was 1892kW with a standard deviation of 312kW. So the IT power consumption was rather stable.

To calculate the partial PUE of the SuperMUC and not the PUE of the LRZ, a slightly different equation than the one defined in equation 2.2 has to be used. The SuperMUC PUE values were calculated by the use of the following equation:

$$PUE = \frac{ITPower + SystemCoolingPower}{ITPower} \tag{4.1}$$

The PUE has a range between 1.06 and 1.35 and follows rather a seasonal than a diurnal pattern. A reason for the low PUE values is the innovative High Temperature Direct Liquid Cooling equipment of the DC. The standard deviation of the PUE is around 0.3 and thus, quite high. Originally, the IT power and PUE data traces contained some missing values, which were estimated by linear interpolation.

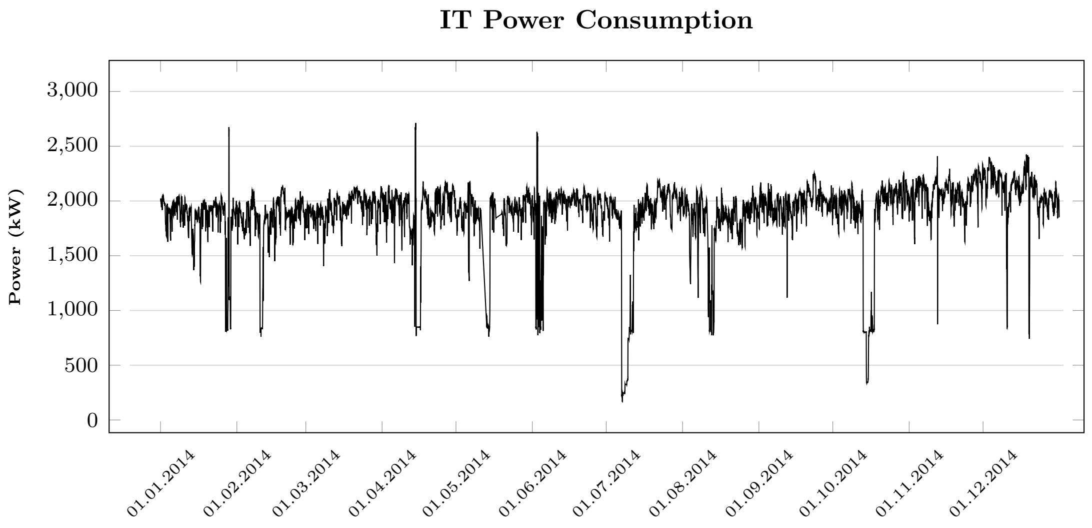


Figure 4.1.: IT power consumption of the SuperMUC.

4.2.3. Derived Data

There are two time series data traces that can be derived from the previously described data. These are *Total Facility Power* and *Cooling Power*. This is only possible under the assumption that the *Total Facility Power* equals the sum of the IT power and the *Cooling Power*. When this assumption holds, then one can utilize equation 4.1 to calculate the *Total Facility Power* from the given IT power and PUE values. The resulting equation would then be: $TotalFacilityPower = PUE * ITPower$. However, when this assumption does not hold, the power that is not directly consumed by the IT equipment (e.g., office spaces, lighting, etc.) is assumed to be part of the *Cooling Power*. As the PUE values in the available partial PUE trace of the SuperMUC are calculated according to Equation 4.1, this assumption holds for the available traces and the obtained *Cooling Power* will actually be the part of the LRZ's *Cooling Power* that was consumed by the SuperMUC. The resulting *Total Facility Power* trace is shown in Figure 4.2. The mean *Total Facility Power* in 2014 was 2131kW with a standard deviation of 342kW. Thus, also the *Total Facility Power* is, as the IT power consumption, quite constant.

Under the same assumptions and by the help of the previously calculated *Total Facility Power* trace, one can use the following equation to calculate the cooling power of the DC: $CoolingPower = TotalFacilityPower - ITPower$. The re-

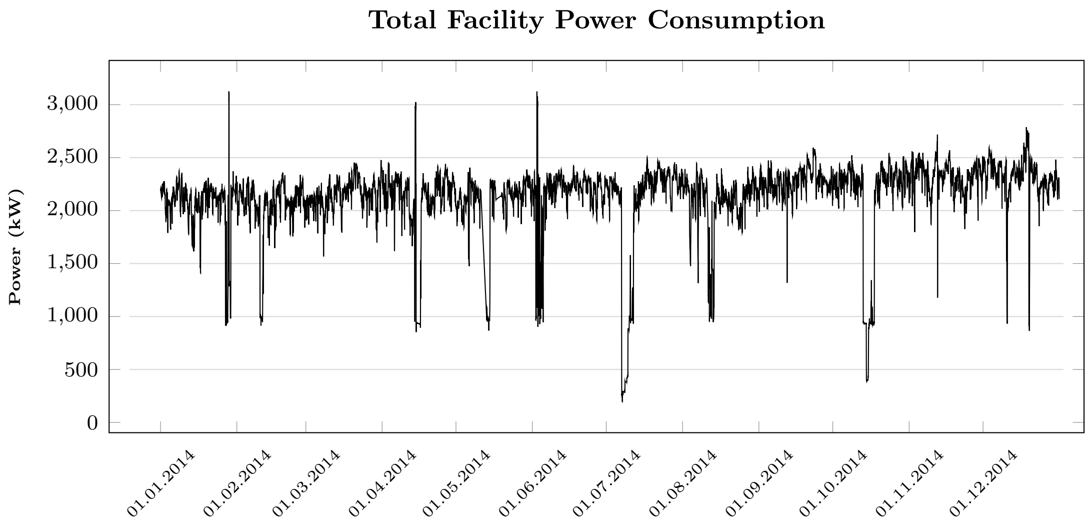


Figure 4.2.: Total facility power consumption that was derived from the SuperMUC traces.

sulting *Cooling Power* data trace has an average cooling power consumption of 239kW with a standard deviation of 58.5. Thus, it is not as stable as the total facility power and the IT power.

4.3. Assumptions

For the simulation scenario the following assumptions are made:

- **A1:** As the SuperMUC executes the workload in a non-virtualized environment, it is assumed that the executed jobs cannot be paused during their execution.
- **A2:** For the workload shifting process, it is assumed that when a job is shifted, the earliest possible restart time of the job is the first timestep after the end of the DR event that caused the shift of the job.
- **A3:** It is assumed that the simulated DC does not utilize predictors for possible changes in the energy price or the amount of workload that will be submitted. Thus, it is assumed that no future knowledge is available for the workload scheduling process.
- **A4:** It is assumed that the simulated DC can implement possible changes of the CPU frequencies in real time without any delay.

5. Analysis of DCSim's Original Status

This chapter describes the original design and implementation details of DCSim, which is the simulator that is utilized for this thesis. Therefore, this chapter is structured into three sections. Section 5.1 describes the role of the framework in the project it was orginally developed for. Section 5.2 describes the original design of the simulation framework. Finally, Section 5.3 provides details on the original implementation of the framework.

5.1. DCSim and All4Green

DCSim is a DC simulation tool which was originally developed to fit into the All4Green framework. All4Green[1] was an EU funded project that intends to optimize energy savings in the complete ecosystem in which DCs operate. The basic architecture of the All4Green framework is illustrated in Figure 5.1. It consists out of 4 layers. On layer 3 several contracts that specify the frame in which the system can operate are located. Layer 2 covers an agent framework, which is used for negotiations between the different actors in the DC ecosystem (DCs, energy providers (EPs), and ITCs). Layer 1 connects the agents on layer 2 to the DC and EP management frameworks, which manage the actual DCs or EPs that are located on layer 0 [BLB+13]. Thus, DCSim is located on layer 0 of the All4Green framework.

Originally, the DCSim framework contained three separate Java projects: *DC-Sim*, *DCSimGUI*, and *DCSimWSClient*. The *DCSim* project contained the actual simulation core, the functionalities needed to simulate a DC, and a web service endpoint via which the simulation can be accessed remotely, the *DCSimGUI* project implemented the basics of a possible graphical user interface (GUI) to the simulator, and the *DCSimWSClient* implemented a web service client that

[1] `http://all4green-project.gfi.es/`

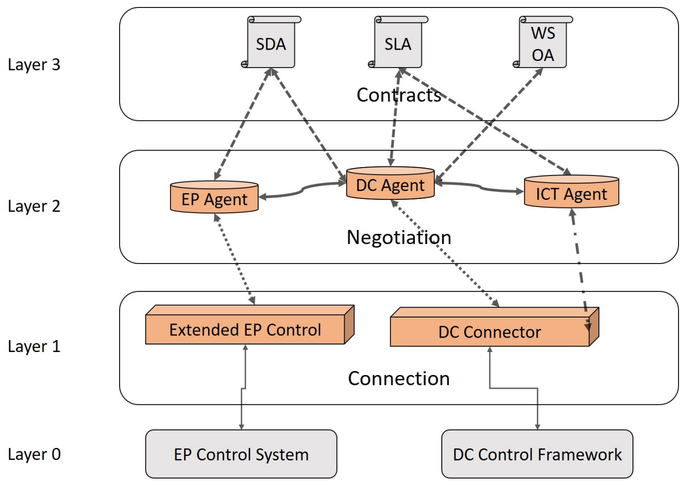


Figure 5.1.: All4Green framework overview, inspired by [BLB+13].

can be used to remotely access a simulation. The web service implementation was necessary due to the role and location of the DCSim simulation tool within the All4Green framework. However, as the the web service and the GUI functionalities are not required for the evaluation of this thesis, the discussion of the current design of DCSim in Section 5.2 will neglect the design parts that are related to the GUI and the web service functionality and focus on the actual simulation core which is implemented in the *DCSim* project and is located on layer 0.

5.2. Original Design of DCSim

DCSim is based on the CloudSim [CRB+11] framework which is an extensible Open Source simulation framework to simulate cloud environments. Both frameworks are developed in Java. CloudSim provides a discrete event simulation core and the basic functionality for the simulation of a DC environment. DCSim extends the CloudSim framework with functionalities that are needed to fit the requirements of the All4Green framework.

In general DCSim is designed to mimic all essential functionalities of a real world DC. Thereby, as the All4Green framework's concern is to optimize energy consumption, the focus lies on the energy consumption of the DC. To address this goal, the simulator models all important energy consumers of a DC. The important energy consumers of a DC, which are simulated by DCSim, can be subdivided into direct and indirect energy consumers. Direct energy consumers include servers, energy storage facilities (ESFs), and the heating, ventilation, and air conditioning (HVAC) system. The most important indirect energy consumer is the executed workload [SKGH12]. Figure 5.2 shows an overview of the original design of the DCSim simulation core.

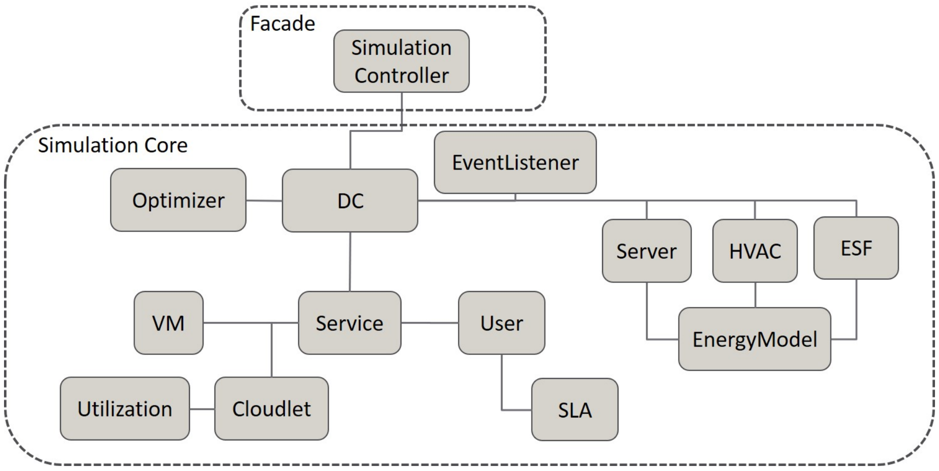


Figure 5.2.: Overview of original design of the simulation framework, inspired by [SKGH12].

Basically, the design can be splitted into two parts: The *Facade* part and the *Simulation Core* part. The *Facade* part contains the **SimulationController** component, which makes the simulation core functionalities accessible to the user. Whereas, the *Simulation Core* part contains the components that are required for the actual DC simulation. One can see that the **DC** component is the central component in the design of the simulator. It is designed to encapsulate all major energy consumer components (**Service**, **Server**, **ESF**, and **HVAC**). Furthermore, it is designed to contain an **EventListener** component and an **Optimizer** component. The **EventListener** component is responsible for the correct reaction to occurring events, whereas the **Optimizer** component is supposed to take care of an optimal resource allocation. The design also requires all active energy consumers to contain an **EnergyModel** component which models the energy consumption of the corresponding component in a realistic manner. The **Service** component is designed so that it can contain one or several **VM** and **Cloudlet** components. Thereby, each **Cloudlet** component can have a different utilization which is modeled by the **Utilization** component. The third component of a **Service** is the **User** component, which is also defines the **SLA** that a certain user has for the usage of a service. To provide a well defined interface to access the monitor data of the simulation, the design defines a **Facade** component.

5.3. Original Implementation of DCSim

To understand which changes to the original version of DCSim are necessary to fit the requirements of this thesis, it is important to know the basic details of the original implementation. Thus, this section intends to introduce the basic implementation details of the original version of DCSim. All classes and resources which are discussed in the remainder of this section are located within the *DCSim* java project. As this thesis focuses on the SuperMUC, which solely executes batch workloads, the implementation details of the service related components are not important for the further understanding of this thesis. Thus, the implementation details of the `Service`, `VM`, and `Optimizer` classes are excluded from this section, but can be found in Appendices A.1, A.2, and A.3. Another component that is excluded from this section is the ESF component. It is excluded because the SuperMUC does not provide such components. Further details on the original implementation of the ESF component can be found in Appendix A.4.

The starting point of a simulation is the class `DCSimCore`. At the beginning of each simulation, the setup parameters are read from a configuration XML file. Within the configuration file all important simulation parameters like the scheduling interval, the simulation speed, the simulation length, the start time of the simulation, and the DC configurations are specified. A DC configuration includes all variables that are necessary to describe a DC. These are a name, the PUE value (it is constant over time), the number of hosted servers, the hardware specifications of each server, the standard temperature of the HVAC system, the specifications of installed ESFs, all services that are offered by the DC with their resource requirements, the default SLA contract definition of the DC, a utilization trace, and some pricing data. It is possible to specify more than one DC. The `DCSimCore` class stores a list of `DC` instances which contains one entry for each DC that is specified in the configuration file. After the initialization process, the `DCSimCore` class handles the flow of the simulation by elapsing the simulation time according to the speed specified in the configuration file. During this process, the `update`-method of the `DC` class is called in regular intervals of the configured scheduling interval length.

To further discuss the basic implementation details of all classes within the framework, the remainder of this section is structured into several subsections. Subsection 5.3.1 describes the implementation details of the `DC` class. Subsequently, Subsection 5.3.2 introduces the basic implementation details of the `HVAC` class which embodies the HVAC component and briefly describes the energy consumption models that are used in the original version of DCSim. The implementation details and the utilized energy consumption models of the second important energy consumer component, which is the server, are provided in Subsection 5.3.3. Finally, Subsection 5.3.4 describes the SLA concept which is used in the original version of DCSim.

5.3.1. DC

The `DC` class represents the DC component within the implementation. Each instance of it stores lists of all servers, services, and monitoring data. The main tasks of the `DC` class are to capture the energy consumption data for later analysis and to react to events that might occur. These tasks are handled by the `update`-method. Within this method, the current overall power consumption, server power consumption, server utilization, HVAC power consumption, HVAC utilization, and ESF power consumption is calculated and stored. Furthermore, it is checked whether there was a change to a service of the DC. When this is the case, an instance of the `Optimizer` class is called to adjust the resource allocation to the new situation. Otherwise, only the `update`-method of each `Service` instance is called. To calculate the total facility power consumption of the DC, the method sums up the power consumptions of all active energy consumers. Each of the consumer classes also provides an `update`-method which is called by the `DC` class each time before the power consumption is requested. This is done to ensure that the power consumption value, which is returned by the instances, is adjusted to the current situation of the simulation.

5.3.2. HVAC

The `HVAC` class stores several attributes that are needed for the implementation of the temperature and HVAC energy consumption models which are utilized in DCSim. These are the current temperature setpoint, the current temperature within the DC building, the minimal reachable temperature, and the temperature change per time interval. Furthermore, the class stores the current power consumption, the current utilization, and two lists to store the historical time series data of these two values. Similar to all other energy consumer classes of the framework, the method that implements the central logic is the `update`-method. The logic of the method adjusts the current building temperature, the current utilization, and the current power consumption to the current simulation status. To do so, it takes the current overall server utilization, the current overall server power consumption, the current PUE, and the scheduling interval as parameters. More information on the exact models that are implemented in the original version of DCSim to determine the changes of the building temperature and the utilization of the HVAC system, can be found in Appendix A.7.

The HVAC energy consumption is modeled by one single model for the whole HVAC infrastructure. DCSim offers one very simple approach and one more complex approach to model the energy consumption of the HVAC system. The simple approach relies on the fact that the main variable heat producers within a DC are the servers. Thus, one can define the variable part of the HVAC's energy consumption as a factor multiplied by the energy consumption of the servers. When the energy consumption of the servers is very large compared to the other consumers (light, network, etc.), this variation is basically captured by the PUE measure. Therefore, one could use the PUE to model the energy consumption of the HVAC system [SKGH12]. When this approach is used, the current power consumption of the HVAC system is calculated via the following equation:

$$p_{HVAC} = p_{servers} * (PUE - 1) * (1 - \frac{t_{building} + t_{min}}{100}) \tag{5.1}$$

Where p_{HVAC} is the current power consumption of the HVAC system, $p_{servers}$ is the current power consumption of all servers of the DC, $t_{building}$ is the current temperature of the DC building, and t_{min} is the minimal temperature inside the DC building. However, the smaller the energy consumption of the servers and consequently also the PUE gets, the less accurate is this approach of modeling the energy consumption of the HVAC system [SKGH12].

The more complex approach tries to include the impact that all other heat producers within the DC have on the energy consumption of the HVAC system. This approach is based on an approach of STULZ [SS11]. In this approach the energy consumption model also takes into account the heat production of other components like other IT components, light, backup generator, USV, utility, condensators, humidifiers, and ac. The original version of DCSim provides four different STULZ HVAC energy consumption models that can be used without further effort. These are a traditional system (PUE around 2.2), a water cooled system (PUE around 3), a direct free air cooled system (PUE around 1.18), and a indirect free air cooled system (PUE around 1.35) [SKGH12]. A detailed description of the implementation details of the STULZ based energy consumption model can be found in Appendix A.7.1.

5.3.3. Server

The `Server` class represents a server within the implementation of the simulator. Within the class, several attributes which are related to the hardware specification of the corresponding server are stored (e.g., size of RAM, harddisk size, network bandwith, etc.). In addition, there are attributes to capture current power consumption, utilization, and time series data for these two values. Another important server specific attribute is a list of virtual machines (represented by instances of the class `VM`) that are currently hosted by the corresponding server. Within the `update`-method of this class, the current power consumption and the utilization are adjusted to the current simulation situation. A detailed description of the procedure that is used to determine the utilization of a server can be found in Appendix A.6.

The energy consumption of a server is empirically modeled for the whole server. Originally, in DCSim a utilization based empirical energy model is used. To create such a model, the energy consumption of a server is measured for eleven different utilization levels (0%, 10%, 20%, ..., 100%). Afterwards, linear interpolation is used to model the energy consumption between these levels. The *eu.a4g.dcSim.powerModels* package of the DCSim project contains several server power models that can be used out of the box. All these power models are inherited from the abstract class `PowerModelSpecPower` which itself implements the `PowerModel` interface. This interface defines only one `getPower`-method, which takes the current server utilization as a parameter. The abstract class `PowerModelSpec-Power` defines an additional abstract method `getPowerData`, which is designed to return one of the eleven measured energy consumption levels. The `getPower`-method of the `PowerModelSpecPower` class implements the linear interpolation approach, where it uses the `getPowerData`-method to retrieve the measured data points.

5.3.4. SLA

The `SLA` class is basically a container class to encapsule all values that are necessary to describe a SLA specification. The original version of DCSim uses a SLA model for interactive workloads. Figure 5.3 shows all metrics that are monitored and used for the SLA model.

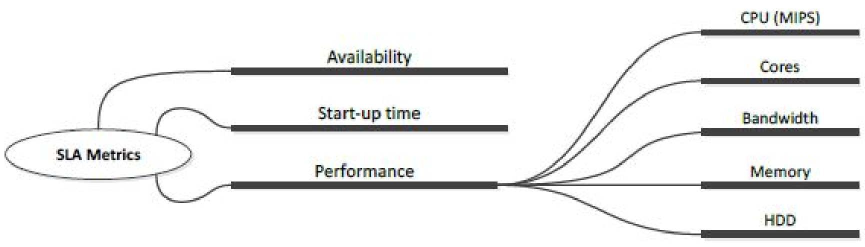


Figure 5.3.: Monitored SLA metrics (taken from [SKGH12].

Each SLA is defined per customer and per service. The performance metrics specify a minimum value that has to be provided to the customer. When one of the performance metrics drops below the contracted value, the DC has to pay a compensation fee. The availability of a service is often expressed in the amount of redundancy that the DC uses to provide the service. Thus, when a high availability is contracted, the DC has to set up many redundant VMs that provide the corresponding service. The start up time quantifies the amount of time that the DC is allowed to take to start a service. The longer this time-span is, the further a service can be shifted backwards in time, which allows the DC to reach higher power demand flexibility.

6. Adjustments to DCSim

As the original version of DCSim only supported the simulation of DCs that execute solely interactive services in a virtualized environment, but the SuperMUC, which is considered for this thesis, only executes batch workload in a non virtualized environment, it was necessary to make some major adjustments to the original design and implementation of DCSim. This chapter intends to describe the adjustments that were made and to explain why they were necessary. Therefore, this chapter is structured into three sections. Section 6.1 highlights the requirements that the adjusted simulation framework has to fulfill. Section 6.2 describes the adjustments that were made to the design of the simulation framework. Subsequently, Section 6.3 introduces the implementation details of the newly introduced components and describes the changes that were made in the implementation of the components that already existed in the original implementation.

6.1. Adjusted Requirements

The general goal of the adjusted version of DCSim is to provide the possibility to simulate the impact of the provision of power demand flexibility and use of different scheduling strategies onto the energy and power use, as well as the energy- and SLA costs of a large DC. To reach this goal and to fit the simulation scenario that is considered in this thesis, the adjusted version of DCSim has to fulfill the following requirements:

- **R1:** To provide the user with the possibility to test several different scheduling strategies, the simulation framework has to provide an easy possibility to exchange the applied strategy and to easily implement new scheduling strategies.
- **R2:** The simulation framework has to be able to simulate the execution of batch workload in a realistic way.

- **R3:** For the realistic simulation of some power demand flexibility provision techniques (e.g., DVFS), the simulation framework has to provide the possibility to model the impact of the application of such techniques onto the runtime and power consumption of the batch workload.
- **R4:** In order to simulate the reaction of the DC to possible DR requests, the simulation framework has to provide a component that handles such requests and provides several possibilities for the evaluation of the impact of the DR request onto the energy- and SLA costs of the DC.
- **R5:** To simulate the energy costs correctly, the simulation framework has to be able use a dynamic energy price.
- **R6:** To fit the considered simulation scenario, the simulation framework has to be able to handle large amounts of compute nodes and workload in a reasonable processing time.

6.2. Adjusted Design

Figure 6.1 shows the schema of the adjusted design of DCSim. The components which are marked in green, are newly introduced in comparison to the original design. One can see that also the adjusted design is basically structured into two parts. These are namely the *Facade* part and the *Simulation Core* part. The *Facade* part contains the SimulationController component which can be used to control a simulation. To monitor important simulation data, the SimulationController is connected to a database. In contrast, the *Simulation Core* part contains all components that form the actual simulation core and thus provide all functionalities that are required for the simulation. The remainder of this section describes the responsibilities of all newly introduced design components in more detail and highlights why these components are required for the adjusted version of DCSim.

BatchJob

As the name already suggests, the **BatchJob** component is designed to represent a single batch job. This component is introduced due to Requirement **R2**.

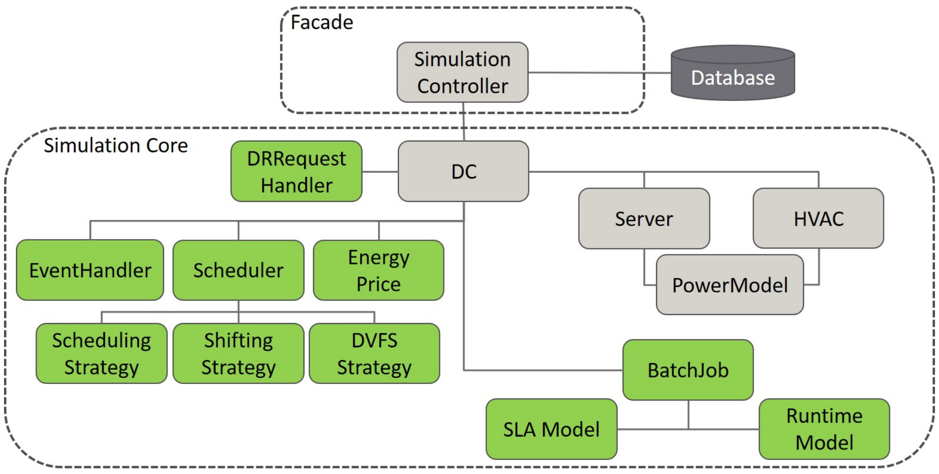


Figure 6.1.: Adjusted Design of DCSim.

From the model in Figure 6.1, one can see that the **BatchJob** component is a direct subcomponent of the **DC** component. Thus, each batch job is exclusively assigned to a distinct DC. The **BatchJob** component has two subcomponents, which are namely the **SLAModel** and **RuntimeModel** components. The **SLAModel** component is also marked in green, because in comparison to the component that existed in the original design, it provides a different type of SLA which fits the common SLA definition of batch workload. The second subcomponent is the **RuntimeModel** component, which is designed to capture a model of the runtime of a batch job. This component addresses Requirement **R3**, because some power demand flexibility provision techniques might not only have an impact onto the power consumption of a server that currently executes a batch job, but also on the time that is needed until a batch job is finished.

EnergyPrice

The **EnergyPrice** component, which addresses Requirement **R5**, is designed to capture the energy price that the DC has to pay for its consumed energy. Depending on the implementation, this price could be static or dynamic over the simulated timespan.

Scheduler

This component is designed to determine the schedule that the execution process of the batch workload follows. The **Scheduler** component is required to fulfill Requirement **R2**.

In the adjusted design, Requirement **R1** is approached by the **SchedulingStrategy** component, which is a direct subcomponent of the **Scheduler** component. The **SchedulingStrategy** component is designed to embody the strategy that is used by the **Scheduler** component to determine the schedule for the workload execution. For the case that a user wants to change the used strategy, the user can simply change this component of the system.

The two other subcomponents **ShiftingStrategy** and **DVFSStrategy** are designed to fulfill similar purposes as the **SchedulingStrategy** component for the strategies that are related to the power demand flexibility provision strategies. The **ShiftingStrategy** component is responsible to determine which jobs will be shifted in the case that workload has to be shifted to increase/reduce the current power demand of the DC. Similarly, the responsibility of the **DVFSStrategy** is to determine which jobs will be affected by the frequency change that is implemented during the application of DVFS to increase/reduce the current power demand of the DC. As for the shifting strategy, the user can simply exchange these two components if the user wants to evaluate the performance of different strategies for workload shifting or DVFS.

EventHandler

As DCSim uses an event based communication mechanism, it has to provide a component that handles the events which occur during the simulation. This task is done by the **EventHandler** component of the system. In addition, this component is also designed to allocate the resources of the DC to the currently scheduled workload. This feature is the main difference between the **EventListener** component in the original design and this component in the adjusted design. The **EventListener** component in the original design was only responsible to listen for events and ensure that a proper reaction is implemented in the system, whereas the resource allocation was mainly done by the **Optimizer** component.

However, in the adjusted version of DCSim, due to Requirement **R2**, there are some new events that did not exist in the original version (e.g., job start, job end). The reaction to these events are closely linked to the resource allocation task within the DC. Therefore, the adjusted design requires these two tasks to be handled by a single component.

DREventHandler

This component addresses Requirement **R4**, it is designed to handle any request that is related to the provision of power demand flexibility of the DC.

6.3. Adjusted Implementation of DCSim

This section describes the adjusted implementation details of the DCSim simulation framework. All components of the DCSim simulation framework are located within the *DCSim* java project.

The starting point of the simulation is the class `Main` which is located in the *simulationControl* package. This class sets the configuration file path of the `DCSimCore` class, which is located in the *simulationControl* package, to the file that is passed as a parameter, starts the simulation core, and finally checks in 100 millisecond intervals whether the simulation has ended or not. In the case that the simulation has ended, the program terminates. The `DCSimCore` class stayed almost unchanged compared to the original implementation (cf. Section 5.3). The major differences are the methods that the class calls for each simulation timestep. The simulation framework uses a discrete simulation time, where one step in simulation time can correspond to an arbitrary period of time in real time. The length of such a simulation timestep can be defined in the configuration. The `update`-method of the original implementation of the `DC` class was splitted into the two methods `scheduleJobs` and `updateJobAllocation`, which are called for every timestep in the adjusted version. This split was done to clearly separate the functionality that is related to the scheduler component and the functionality which is concerned with the event handling and resource allocation. More information on these two methods are provided in Subsection 6.3.2. In addition, the class now ensures that all data values, which are important for the evaluation, are stored in a SQLite[1] database. The structure of the *.xml* configuration file which is parsed at the start of the simulator was adjusted to fit the needs of the adjusted version. A document type description of the required structure for the configuration file is provided in Appendix B.1. Besides file paths of required data traces (e.g., workload trace), there are four additional configuration parameters that were added to the configuration file. These are the amount of real time seconds that a simulation timestep lasts, a `boolean` value that indicates whether the simulator runs in *superMUCMode* or not, and a solver timeout value. The two latter named values are used during the scheduling process of the simulator and are explained in more detail in Subsection 6.3.5. In the remainder of this section, the implementation details of all system components and the power consumption models which are used in the adjusted version of DCSim are explained in more detail.

6.3.1. Batch Job Concept

The batch job concept is basically implemented by the class `BatchJob` and the enum `BatchJobStatus` which are both located in the *utilities* package. The `BatchJobStatus` enum defines the different statuses that a batch job can have within the system. These are `PARSED`, `SUBMITTED`, `SCHEDULED`, `RESCHEDULED`, `RUNNING`, `PAUSED`, and `FINISHED`. Figure 6.2 illustrates the possible statuses a

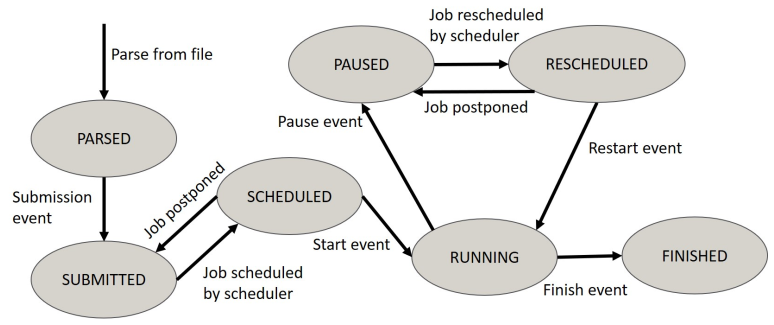


Figure 6.2.: Possible transitions for the status of a batch job.

batch job can have and when it transitions from one status to another. Initially, when a batch job is read from the workload trace file, it will take the `PARSED` status. Afterwards, when the submission event of a batch job was handled by

[1]`https://www.sqlite.org/index.html`

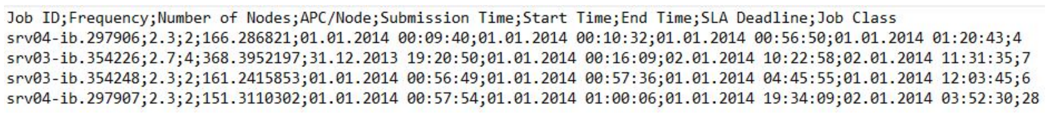
the event handler component, the job will take the status `SUBMITTED` and the scheduler will be able to schedule it. Once the scheduler component has scheduled a batch job, it will transition to the `SCHEDULED` status. From the `SCHEDULED` status, a job can either transition back to the `SUBMITTED` status, which happens when it is postponed due to an DR event, or, in the case that the event handler processes the start event of the job, to the `RUNNING` status. From this status, the job can either transition to the `PAUSED` status, in the case that a pause event is issued for the job, or to the `FINISHED` status, which will happen in the case that the finish event of a job is processed. When a job is in the `PAUSED` status, the scheduler can reschedule the job again which will lead to a transition to the `RESCHEDULED` status. From the `RESCHEDULED` status, the job can either transition back to the `PAUSED` status (which happens when the job is postponed due to an DR event) or, if the restart event is processed, it transitions to the `RUNNING` status.

The actual `BatchJob` class stores several attributes that describe all relevant attributes of a batch job. The most important ones are an id, the planned CPU frequency for execution, the execution time that a job needs, the number of compute nodes that should be used for the execution, the average power consumption of a job, the SLA deadline, the finish delay, and the current status of a job. Furthermore, the class stores the submission time, the planned start time (only initially exists when it is specified in the job data trace, otherwise this is set by the scheduler), and a possible restart time (only exists when the job got paused during its execution). All these times are stored in simulation time. In addition, the class stores three events that correspond to the start, finish, and possible restart events of the job. This is necessary in order to ensure that at any point in time only one of the named events exist in the system for each job. More details on the different types of events and the event handling can be found in Subsection 6.3.4 and Appendix B.2. Every instance of the class also stores a list of the servers that are currently assigned to the job. This list is empty when the corresponding batch job is currently not in the status `RUNNING`. The server list is mostly used for the resource allocation process, which is handled by the event handler component.

To model the SLA cost of a job correctly, the job also stores an instance of the `SLAModel` interface, which is located in the *SLAModels* package. This interface defines two methods that are designed to provide information that are required to calculate the SLA cost of a batch job. More information on the model that is implemented within the adjusted version of the simulation framework can be found in Subsection 7.4. The SLA cost of a job can be retrieved via the `calculateSLACosts`-method of the `BatchJob` class. This method takes one parameter as input which corresponds to the usage price that the simulated DC charges per compute node hour.

To model the remaining runtime of a batch job correctly and to be able to calculate the estimated finish time, the `BatchJob` class also contains three attributes that contain the elapsed runtime, the remaining runtime, and the paused time. All these three variables are specified in simulation time. These variables are also required in order to model the impact of some power demand flexibility provision techniques onto the remaining runtime of a job. Therefore, the class also has an attribute that stores the runtime model of the batch job, which is used to recalculate the remaining runtime in case of a frequency adjustment. During the simulation the remaining time, elapsed time, and paused time can be updated via the `elapseOneSimulationTimestep`-method, which takes different actions depending on the current status of the batch job. The estimated finish time of a job can be retrieved via the `getCalculatedFinishTime`-method of the class. When the job currently is rescheduled, the estimated finish time is calculated as the sum of the scheduled restart time and the remaining runtime. Otherwise, the estimated finish time is calculated as $estimatedFinishTime = t_{start} + t_{elapsed} + t_{remaining} + t_{paused}$, where t_{start} is the start time of the job, $t_{elapsed}$ is the elapsed runtime of the job, $t_{remaining}$ is the remaining runtime of the job, and t_{paused} is the time that the job spent in the `PAUSED` status.

The batch workload trace for the simulation is read from a *.csv* file during the setup phase. This is done by the `BatchJobParser` class, which is also located in the *utilities* package. The required structure of the *.csv* file depends on whether the *superMUCMode* is used or not. When the mode is used, the simulator tries to exactly rebuild the job power profile and thus, also the schedule that was executed by the SuperMUC in 2014. To do so the input file has to provide a job id, the planned execution frequency, the number of utilized compute nodes, the average power consumption per node, the submission time, the start time, the end time, the SLA deadline, and the job class value of the job in exactly this order. Some example entries for such a file are illustrated in Figure 6.3. The submission, start,



```
Job ID;Frequency;Number of Nodes;APC/Node;Submission Time;Start Time;End Time;SLA Deadline;Job Class
srv04-ib.297906;2.3;2;166.286821;01.01.2014 00:09:40;01.01.2014 00:10:32;01.01.2014 00:56:50;01.01.2014 01:20:43;4
srv03-ib.354226;2.7;4;368.3952197;31.12.2013 19:20:50;01.01.2014 00:16:09;02.01.2014 10:22:58;02.01.2014 11:31:35;7
srv03-ib.354248;2.3;2;161.2415853;01.01.2014 00:56:49;01.01.2014 00:57:36;01.01.2014 04:45:55;01.01.2014 12:03:45;6
srv04-ib.297907;2.3;2;151.3110302;01.01.2014 00:57:54;01.01.2014 01:00:06;01.01.2014 19:34:09;02.01.2014 03:52:30;28
```

Figure 6.3.: Example entries of the workload trace input file when *superMUC-Mode* is switched on.

and end times and the SLA deadline can be either specified in simulation time or as date strings that represent the corresponding times in real time. When date strings are used to specify the times, the parser will automatically convert the dates into simulation time. In both cases, the parser determines the duration of the job in simulation time on the basis of the specified start- and end time. The difference in the case that *superMUCMode* is switched off, is that the simulated DC will use an own scheduling strategy to schedule the jobs and will not try to replicate the exact schedule of a real world DC. Thus, the structure of the input file slightly changes in this case as it only needs to provide a job id, the planned execution frequency, the number of utilized compute nodes, the average power consumption per node, the submission time, the duration, the SLA deadline, and the job class of the job. Also in this case, the information have to be provided in exactly this order. The job class value of the job records is related to the server power model which is currently implemented. More information on this can be found in chapter 7.

6.3.2. Data Centre

Basically, the adjusted version of the `DC` class maintains the same configuration parameters as the original version. The main differences are related to the introduction of the batch job concept and the event handling within the simulation framework. Due to Requirement **R6** (c.f., Section 6.1), the amount of values that are directly stored within the class for monitoring purposes is reduced. Otherwise, the size of the previously stored value lists get very large with increasing simulation time and thus cause an immense increase of the processing time of the simulation framework. Instead, all relevant values are now maintained within a SQLite database which is much more efficient. However, the class still maintains variables that capture the current total power consumption of the DC, the current power consumption of the HVAC infrastructure, and the current power consumption of all servers.

Due to the introduction of the batch job concept, the adjusted version of this class now maintains lists of batch jobs instead of a list of services. Within one instance of the class, which represents one DC, two separate lists of batch jobs are maintained. One list contains all jobs that currently have the status `PARSED` and the other list contains all jobs that currently have the status `FINISHED`. This split is mainly necessary to enable other methods to only loop through the jobs that are relevant for them, which can reduce the runtime of one simulation run significantly. Besides the lists of jobs, instances of the adjusted `DC` contain additional variables and methods that are related to the scheduling and event handling within the simulation framework.

Another major difference is that the `update`-method was splitted into the `schedule-Jobs`- and `updateJobAllocation`-methods. This is mainly due to the introduction of the batch job concept, because this concept requires a scheduler component that is usually not active during all timesteps, but only at the beginning of a scheduling interval. A scheduling interval defines a period of time for which the workload is scheduled during one call of the scheduler. The length of these intervals can be configured in the configuration file. Both methods are called for each simulation timestep within the `DCSimCore` class, as it is possible that the length of the scheduling interval equals the length of a simulation timestep. Therefore, the `scheduleJobs`-method, which activates the scheduler component, has to ensure that the scheduler component of the DC is only called when the current timestep actually equals the start of a scheduling interval. The basic structure of the `updateJobAllocation`-method is similar to the `update`-method in the original implementation. The two main differences is the handling of the events and how the current server power consumption is calculated. At the beginning of the method, the event handler of the DC is called to handle all events that are scheduled for the current timestep. The only events that are not handled by the event handler itself, but are handled directly in the `updateJobAllocation` method, are `SERVER_UPDATE` events. These events are always scheduled when the current status or power consumption of a server has changed. For each scheduled

SERVER_UPDATE event, the method calls the update-method of the corresponding Server instance and adjusts the total current power consumption sum of all servers. More information on why the event handling was adjusted, the event handling procedure in general, and the different types of events can be found in Subsection 6.3.4 and Appendix B.2. When all SERVER_UPDATE events are processed, the method calculates the total current power consumption of the DC as the sum of the current server power consumption and the HVAC power consumption and sets the current simulation time clock to the next step.

6.3.3. Server and HVAC

The implementation of the HVAC class stayed almost untouched compared to the original version of DCSim. The only difference is that it now uses a power consumption model that calculates the current power consumption on the basis of the IT power and the PUE.

In the Server class slight changes were necessary to support the newly introduced batch job concept. To support the new concept, the class now has a variable that stores an instance of the BatchJob class, which represents the batch job that is currently running on the server. In addition, one public constants was added that specifies the power consumption of the server in the idle state. The third major change is the adjustment of the update-method which now uses a frequency based power consumption model to determine the current power consumption of the server. The current power consumption of the server is determined by the call of the getServerPower-method of the PowerModelSelector class, which is located in the *powerModels* package. The PowerModelSelector class is designed to provide the user a very convenient way to exchange the power model that is used. To change either the server power model or the HVAC power model, the user can simply change the specified power model for each of the two components within the PowerModelSelector class. Another minor change is that the status of the server is not longer captured by a string variable but is now captured by the newly defined ServerStatus enum, which is located in the *utilities* package. The possible server statuses are OCCUPIED, IDLE, and OFF. However, in the current implementation the OFF status is never used.

6.3.4. Event Handling

The event handling was adjusted compared to the original implementation mainly due to Requirement **R6** (cf. Section 6.1). Another minor reason is that the event handling was very inflexible in the original version as it was not possible to schedule events at a specific point in simulation time but only via the `isChanged`-flag within certain simulation core components. In the original version the events were then detected and handled by looping through all component instances and check for each instance whether the `isChanged` flag is set or not. In the adjusted version this is infeasible, because it has to handle a much larger amount of workload and servers as well as a longer simulation timespan (the data that are provided by the DC, which is considered for this thesis, span over one entire year). As the old event handling would have required a check of every batch job for every simulation timestep, for a simulation with a workload trace which contains 400000 jobs that are executed on a DC with 9216 compute nodes and a simulated timespan equals one year this would have resulted in roughly $400000 * 9216 * 31536000 \approx 1.16 * 10^{17}$ loops just for the event handling. This would lead to a very long execution time for one single simulation run. Nevertheless, as it is very unlikely that all jobs and all servers trigger an event during each timestep in the simulation time, there is a large potential to reduce the amount of executed loops. When one for example would know for which servers and batch jobs the `isChanged` flag is set upfront, one could only loop through the relevant servers and batch jobs. As the flag has to be set by some other component that detected that an event has to be triggered, it is possible to have this knowledge upfront. To tackle this, in the adjusted version of DCSim, an event queue was introduced that maintains a list of events for every timestep at which at least one event is scheduled. Therefore, in the adjusted version, events are scheduled by adding an event to this event queue instead of setting the `isChanged` flag within a server or batch job instance. In addition, this approach enables the possibility to introduce additional, more complex events and provides more flexibility for the scheduling of events. The adjusted version of DCSim uses six different types of events, which are namely `JOB_START`, `JOB_FINISH`, `JOB_PAUSE`, `JOB_RESTART`, `JOB_SUBMISSION`, `DR_REQUEST`, and `SERVER_UPDATE`. Additional details on the implementation of the event handling and resource allocation process are given in Appendix B.2.

6.3.5. Scheduling

The main part of the logic for the scheduler component of the simulation framework is implemented in the `Scheduler` class. Within the class, two variables of the types `DemandFlexibilitySchedulingStrategy` and `SchedulingStrategy` contain the instances of the classes that implement strategies for the power demand flexibility provision and the scheduling. More details on these classes and the interfaces that they implement are described later in this subsection. Furthermore, the class maintains three different lists of batch jobs. One list that contains all scheduled jobs, one list that contains all jobs that were already submitted to the scheduler, and one list (`affectedSubmittedJobs`) that only is used when the simulator runs in *superMUCMode*. This list contains all jobs for which it was not possible to schedule or start them at the original start time that is specified in the workload trace. That might happen due to DR events during which workload that originally was executed earlier was postponed, or because the start times in the workload trace are not correct and it happens that the start of the job would increase the amount of occupied compute nodes above the number of available compute nodes. The jobs are maintained in separate lists in order to improve the performance of the simulation framework as it is then possible for a method to only loop through a part of the jobs that is actually relevant for the operation of the method.

The central method of the `Scheduler` class is the `scheduleNextInterval`-method which is designed to schedule the jobs for the upcoming scheduling interval. Both, the `DC` class and the `DRRequestHandler` class, which is described in subsection 6.3.6, use this method to schedule the upcoming interval. Thus, the method has to provide the possibility to specify the parameters of a possible DR request. Within this thesis a DR request is characterized by three parameters. These are the adjustment height, the adjustment direction, and the adjustment length. The meaning of the three parameters is illustrated in Figure 6.4. Within the figure one can see a small part of the total facility power profile of the SuperMUC, where the x-axis is the time axis and the y-axis is the power consumption axis. The red square in the right half of the figure defines the timespan where a DR event was issued by the grid operator of the DC. In this case, the adjustment length would be equal to the length that the red square has in x-axis direction.

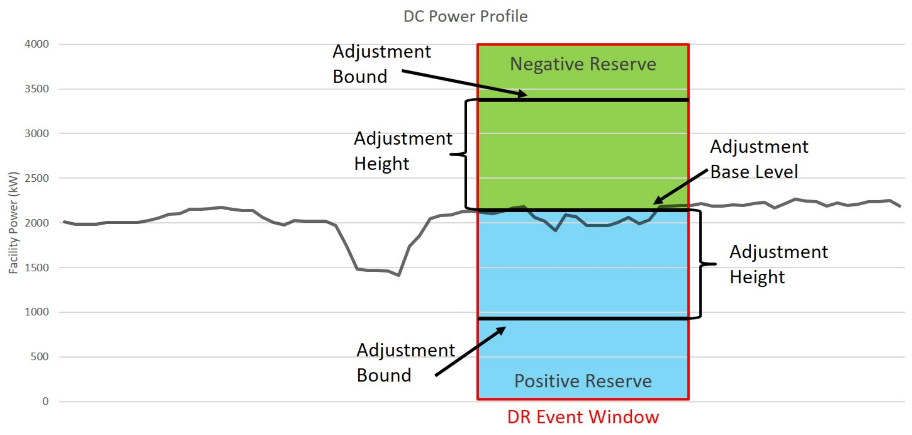


Figure 6.4.: DR request concept that is used in this thesis.

The height and the direction of the adjustment is determined based on the power consumption level that the DC has at the beginning of the DR event. In the figure, this basis level is indicated by the solid black line that is located in the center of the red square. Depending on the position of the basis level, the adjustment height specifies at which position on the y-axis the adjustment bound is placed. Whether the adjusted power profile of the DC has to be above or below the adjustment bound for the entire length of the DR event window, depends on the adjustment direction. The adjustment direction is also expressed in the terms positive reserve provision or negative reserve provision. In Figure 6.4, the green shaded area indicates the area in which the DC would provide negative reserve power and the blue shaded area indicates the area in which the DC would provide positive reserve power. Thus, in the case of negative power provision, the DC has to increase its power consumption and the adjusted power profile has to stay above the adjustment bound during the DR event window. In contrast, when the DC provides positive reserve power it has to decrease its power consumption and the adjusted power profile has to stay below the adjustment bound during the DR event window.

The `scheduleNextInterval`-method takes the adjustment height in watts, the adjustment direction as a value of the `ReserveProvisionType` enum, and the adjustment length in simulation time as parameters. When the method is used to schedule an interval which is no DR event interval, the adjustment length parameter is used to specify the scheduling interval length and the adjustment height parameter is set to 0. The `ReserveProvisionType` only defines two constants, which are the two reserve provision types `POSITIVE` and `NEGATIVE`. At first, depending on the specified provision type, the method calculates the consumption bound that has to be fulfilled by the adjusted power profile of the DC. The scheduling procedure, which happens afterwards, depends on whether the simulator runs in *superMUCMode* or not:

superMUCMode on: At first, it is tried to schedule all jobs that were shifted during a DR event or were not started at the original start time that was specified in the data trace (jobs that are contained in the `affectedSubmittedJobs` list). To schedule these jobs, the scheduler uses a scheduling strategy which can be implemented by any class that implements the `SchedulingStrategy` interface. More information on this interface and the scheduling strategy, which is currently implemented, is given later in this section. Subsequently, the scheduler tries to schedule all jobs, which were originally started in the current scheduling interval, at their original start time. When this is not possible for a job (for example because there are not enough nodes available), the scheduler checks whether the job can be scheduled at any later timestep within the current scheduling interval. Should that not be possible as well, the job is added to the list that contains all jobs that could not be started at their original start time (`affectedSubmittedJobs`). During this procedure, the scheduler uses backfilling. This means that in the case that a job X cannot be scheduled, afterwards, only jobs that are estimated to finish before the earliest possible start time of X are scheduled.

superMUCMode off: In this case, the scheduler schedules all jobs according to the scheduling strategy that is only used for the already delayed jobs when the simulator runs in *superMUCMode*.

Finally, if the specified adjustment height is larger than 0, it is checked whether the current schedule fits the calculated consumption bound. When this is not the case, the scheduler uses the available power demand flexibility provision techniques to adjust the power consumption in a way so that it fits the calculated consumption bound. The strategies for the usage of these techniques is implemented in the `DemandFlexibilitySchedulingStrategy` component of the scheduler. More details on this component are given later in this subsection.

At any time of the simulation, the current schedule is expressed implicitly by the start/restart times and start/restart events which are stored in the `BatchJob` instances that represent the scheduled batch jobs. Thus, in order to change the schedule, the start/restart times and start/restart events of the `BatchJob` instances have to be changed directly.

Scheduling Strategy

The `SchedulingStrategy` interface, which is located in the *scheduling.schedulingStrategies* package, is designed to provide a well defined interface that is used by the scheduler component to access an implementation of a scheduling strategy. Thus, all classes that should be usable as scheduling strategy within the simulation framework have to implement this interface. The interface defines the `scheduleNextInterval`-method which is designed to schedule the jobs for the upcoming scheduling interval This architecture allows future developers to easily exchange the scheduling strategy that is used for the simulation.

Currently there is one scheduling strategy implemented by the `ScheduleShortestTimeToDeadlineFirst` class, which is located in the *scheduling.schedulingStrategies* package. The implemented strategy is a heuristic based strategy, which uses a shortest time to deadline first (STDF) heuristic to determine in which order the jobs are scheduled. The concept of this heuristic is illustrated in Figure 6.5. In

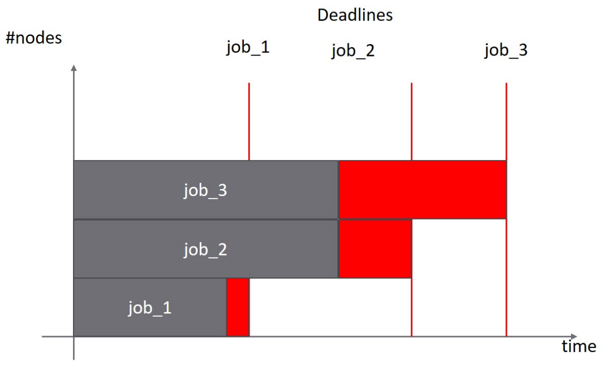


Figure 6.5.: Concept of the STDF heuristic.

the figure one can see three jobs that are assumed to be currently available for scheduling. Each job is represented by a grey box, where the length of the box in x-axis direction indicates the amount of time that each job needs to be finished and the height of each box in y-axis direction indicates the amount of nodes that each job occupies. Each of the three perpendicular red lines indicates a SLA deadline of one of the jobs and the red boxes behind each of the jobs indicate the length of the current time difference between the estimated finish time of a job and its SLA deadline. When a DC uses the STDF heuristic to determine the order in which the jobs are scheduled, the jobs are first ordered, in ascending order, by their Θ_{STDF} values, which are calculated as follows:

$$\Theta_{STDF}(x) = \frac{t_{SLADeadline}(x) - t_{estimatedFinish}(x)}{numberOfNodes(x)} \tag{6.1}$$

Where x is a batch job, $t_{SLADeadline}(x)$ is the SLA deadline of x, $t_{estimatedFinish}(x)$ is the estimated finish time of x, and $numberOfNodes(x)$ indicates the number of nodes that x utilizes. This corresponds to the length of the red boxes in x-axis direction divided by the height of the red box in y-axis direction. Thus, shortest time to deadline does not correspond to the difference between the time of the scheduling and the SLA deadline, but depends on the amount of timesteps a job can be delayed without pushing the finish time of the job behind the SLA deadline. The idea behind this heuristic is to minimize the SLA cost that are caused when it is not possible to schedule all jobs. Therefore, when a DC would schedule the workload, which is illustrated in Figure 6.5, according to the STDF heuristic, it would first schedule job_1, then job_2, and finally job_3.

The implemented strategy tries to schedule all jobs that are in the `SUBMITTED` or `PAUSED` status at the point of scheduling, in the order that is determined according to the STDF heuristic. When it happens that a job cannot be scheduled (for example because there are not enough idle nodes available), backfilling is used for all following jobs.

Power Demand Flexibility Provision Strategy

Within the `DemandFlexibilitySchedulingStrategy` class, the management of the power demand flexibility provision techniques is implemented. In the current version of the simulation framework the available techniques are workload shifting and DVFS. The `adjustPowerConsumption`-method of this component is called by the scheduler component in the case that an adjustment of the power profile is necessary (e.g., due to a received DR request). As an reaction to this call, the currently configured combination of power demand flexibility provision techniques is used to adjust the power profile of the DC according to the received DR request. Such a configuration is expressed as a tuple out of the percentage of shiftable workload that should be shifted and a frequency value to which the workload that remains in the DR event window should be scaled. Thereby, as the length and amount of utilized nodes can vary heavily among the jobs, it is not very accurate to shift the specified fraction of the total number of jobs in the DR event window. In this case the actual shifted amount of power would heavily depend on the workload structure and the utilized shifting strategy. To overcome this problem, the current implementation uses the *nodeSteps* measure to calculate how much workload is supposed to be shifted. The *nodeSteps* measure of a job within a DR event window w is calculated as follows:

$$nodeSteps_w(x) = t_w(x) * numberOfNodes(x) \tag{6.2}$$

Where x is a batch job that is partially or fully scheduled within the DR event window w and $t_w(x)$ indicates the runtime of a job within DR event window w in simulation time. The *nodeStep* measure for a complete DR event window w can be calculated as follows:

$$nodeSteps_w(X) = \sum_{x \in X} nodeSteps_w(x) \tag{6.3}$$

Where X is the set of all jobs that are partially or fully scheduled in w. The amount of workload that is actually supposed to be shifted in DR event window w is then calculated as the specified fraction of $nodeSteps_w(X)$. This measure is suited better than the number of shifted jobs to express the amount of shifted workload, because it correlates better with the actually shifted amount of power.

To actually shift and scale the workload, the `ShiftingStrategy` and `DVFS-Strategy` components of the class are used. These components are described in more detail later in this subsection. When the schedule and the frequencies were adjusted, in the case of positive reserve power provision, the class creates a linear optimization problem to try to maximize the power demand flexibility that can be provided through an optimal scheduling of the workload that remains in the DR event window. In the case of positive reserve provision, an optimization problem with the following structure is created:

$$\begin{aligned} \text{minimize} \quad & z \\ \text{subject to} \quad & -z + \sum_{j=1}^{J} x_{ji} * p_j \leq \theta_i,\ 1 \leq i \leq T \\ & -y_j + \sum_{i=1}^{T} x_{ji} = 0,\ 1 \leq j \leq J \end{aligned}$$

Where z is the minimum power consumption sum out of all timesteps for the optimized DR event window, T is the total amount of simulation timesteps in the DR event window, J is the total amount of jobs that run at least partially in the DR event window, y_j is the amount of simulation time timesteps a job j is supposed to run in the DR event window, p_j is the average power consumption of job j, θ_i is the total power consumption of all jobs that are not shiftable at timestep i, and x_{ji} is a binary variable that indicates whether job j runs at the ith timestep of the DR event window. Assumption **A1** (cf. Section 4.3) causes that jobs which are already running at the start of the interval and jobs which start during the interval, but finish after the interval, cannot be shifted. The first constraint ensures that the objective that is minimized is actually the maximum power consumption sum out of the power consumption sums at each timestep within the DR event window. Whereas, the second constraint ensures that each job in the optimized schedule runs for the same amount of timesteps as in the original schedule. Additional constraints, which are not shown here that ensure that each job is executed as a block (to fulfill Assumption **A1**) and that the amount of occupied nodes does not exceed 9216 at any point in time, are added to each problem.

Depending on the DR event window length, the workload structure and the granularity of the simulation time, the created problem can get very complex and thus, the solving times can become very long. To prevent that the simulator gets stuck in a too complex optimization problem, a *solverTimeout* parameter, which is specified in seconds, was added to the configuration file. When the solver exceeds this timeout it will automatically stop. In such a case a possibly found suboptimal schedule or, if no feasible solution was found so far, the original schedule will be used. Due to the same problem, the solver automatically converts the simulation time into minutes, when 1 second is used for the *secondsPerSimulationTimestep* parameter. To solve the linear optimization problems the java API of the *lp_solve* framework [BEN04], which is an open source linear programming system, is used. When the schedule of the remaining workload was optimized, the `adjustPowerConsumption`-method checks whether the adjusted schedule fits the requested power consumption bound. For the case that the check holds true, the method returns `true` in order to indicate that the requested DR event can be handled successfully. In the opposite case, `false` is returned, as the requested DR event cannot be handled with the currently configured combination of power demand provision techniques.

Shifting Strategy

Any class that implements a shifting strategy within the simulation framework should implement the `ShiftingStrategy` interface, which is designed to provide a well defined interface for the interaction between a shifting strategy and the `DemandFlexibilitySchedulingStrategy` component of the system. In the adjusted version of the simulation framework there is one shifting strategy implemented by the `ShiftLongestTimeToDeadlineFirstStrategy` class, which also uses the STDF heuristic to determine the order in which the jobs should be shifted. However, in the case of a request of positive power reserve provision, the strategy orders the jobs not in ascending but in descending order. This minor adjustment is necessary as the jobs that are shifted first are executed after the jobs that are not shifted and thus, to minimize the SLA costs, the $\Theta_S TDF$ values the jobs that are shifted first have to be larger than of the jobs that are shifted afterwards. Thus, if it is assumed that three jobs are currently scheduled as illustrated in Figure 6.5, the strategy would first postpone job_3, at second position it would postpone job_2, and finally it would shift job_1. In contrast, when jobs are preponed, the shifting strategy determines the order in which the jobs are preponed in the same way as the `ShortestTimeToDeadlineFirst` scheduling strategy.

The assumptions that are made for the simulation scenario in this thesis (cf.

Section 4.3) have some effects on the shifting procedure. Assumption **A1** is not realistic for DCs that execute virtualized workload by the use of virtual machines. However, for the SuperMUC, which is considered in this thesis, does only execute non-virtualized workload this assumption is realistic. Assumption **A2** leads to a small difference between the amount of workload that is requested to be postponed and the amount of workload that is actually postponed. Depending on the current workload structure and the utilized scheduling strategy, this difference might be smaller or larger. This is, because through the second assumption only the complete amount of node steps that a job has in the DR event interval can be shifted out of the interval. Thus, only a few discrete fractions of the total postponable node steps can actually be reached. The size of the steps between these fractions depends on the workload structure.

It is important to note that when the simulator runs in *superMUCMode* and during the postpone process a `SCHEDULED` job is postponed, it cannot be started at its originally scheduled start time anymore. Thus, it is moved to the `affected-SubmittedJobs` list, which is maintained by the scheduler.

DVFS Strategy

Any class that implements a DVFS strategy within the simulation framework should implement the `DVFSStrategy` interface, which is designed to provide a well defined interface for the interaction between a DVFS strategy and the `Demand-FlexibilitySchedulingStrategy` component of the system. The interface can be found in the *scheduling.schedulingStrategies.dvfsStrategies* package.

In the current version of the simulation framework there is one DVFS strategy implemented by the `ScaleAllJobsStrategy` class. As the name suggests, this strategy simply scales all jobs to the requested frequency. After a possible DR event, all jobs are scaled back to their originally specified CPU execution frequency.

6.3.6. DR Event Handling

The DR event handler component of the DC is designed to take possible DR event requests to the DC as an input and to ensure that the DC's reaction is appropriate. Within the adjusted version of DCSim, this component is implemented in the `SimpleDREventHandler` class. This class implements the `issueDemand-ResponseRequest`-method which can be used to issue a DR event request to the simulation. When the method is called, it first checks whether the requested adjustment height is above the maximum technically achievable adjustment height. The maximum achievable adjustment height is reached by shifting the entire affected workload out of the DR event window and scaling the remaining workload to the minimum frequency. In the case that the check does not fail, the scheduler of the DC is requested to schedule the DR event window according to the parameters of the DR event. When it turns out that the original schedule can be adjusted in a way that it fits the requirements of the DR event window, it is ensured that the scheduler is not called again before the end of the DR event window.

The class also provides the possibility to optimize the configuration of the power demand flexibility techniques that are used to react to a DR event. This is done by the `optimizePowerDemandFlexibility`-method. Thereby, the configuration that causes minimal energy- and SLA costs is considered to be optimal, where for the calculation of the energy costs only the power consumption of the single jobs, which are affected by the demand flexibility provision actions, is considered. The additional costs of a configuration are calculated by the `determinePower-DemandFlexibilityCost` with respect to Assumption **A3** (cf. Section 4.3). To calculate the impact on the SLA costs, the additional delay, which might occur due to postponing or a decrease of the frequency, is calculated for each job that is affected by any power demand flexibility technique. As there is currently no predictor for the energy price, the energy price of the timestep at which the DR request is issued is taken to calculate the impact on the energy consumption cost of a job. Therefore, the energy costs of a job will only change when it is scaled, but not when it is shifted. Due to Assumption **A2**, the actually reachable amounts of shifted node steps is a discrete, step-wise function. The optimization procedure evaluates the combination of each of the discrete steps with all possible scaling frequencies, in order to determine the combination that fulfills the requirements of the DR event and causes minimal costs.

To provide the possibility to define a DR event trace which contains several DR events that should be issued during the simulation, the *DREventTraceFile* parameter was added to the configuration. This parameters specifies the path of a *.csv* file that contains the DR event trace. Each record of this trace has to specify all information that are required to define a DR event, which are the time of the start of the DR event, the required adjustment height (in W), the provision type (positive or negative), the length of the DR event (in simulation time), and the compensation that is provided by the grid operator. For all events that are defined in this trace, the simulator will try to optimize the combination first and then issue the DR event request with the optimal strategy combination, if such a combination is available.

6.3.7. EnergyPrice and PUE

To calculate the energy cost of the DC, an energy price is needed. The adjusted simulation framework provides the possibility to use a dynamic energy price trace for this. The trace has to be defined in a *.csv* file stored at the location that is described by the path which is specified by the *EnergyPriceTraceFile* parameter of the configuration. Each record of the energy price trace file has to define a timestep (in simulation time or as a date string) and an energy price in €/kWh. Thereby, an energy price is used from the time on that is defined in the trace until the next timestep, for which an entry exists in the trace, is reached.

Similarly, the PUE trace, which is supposed to be used for the simulation, can be defined. The corresponding file path has to be specified by the *PUETraceFile* parameter of the configuration. The structure of the records of the PUE trace are similar to the ones of the energy price trace, where the energy prices are replaced by the PUE values. As for the energy prices, a PUE value is used from the time that is specified in its record until the next timestep for which a new PUE value is defined is reached.

7. Implemented Models

This chapter introduces all models which are implemented in the adjusted version of DCSim. Therefore it is structured into four sections. Section 7.1 describes the implemented server power consumption model. Section 7.2 introduces to the models that are used to model the runtime behaviour of a batch job under frequency scaling. Section 7.3 describes the estimation approaches that are used to estimate the IT- and cooling power consumption. Finally, Section 7.4 illustrates the SLA model that is utilized in this thesis.

7.1. Server Power Model

Dayarathna *et al.* [DWF16], provided a well structured overview of the the existing power models in the DC environment. However, for this thesis only the kind of models that model the power consumption of an entire server are relevant. According to Dayarathna *et al.*, these kind of models can be basically partitioned into additive models and utilization based models [DWF16]. One very simple additive model was introduced by Tudor and Teo [TT13], which is formulated as follows: $E = E_{CPU} + E_M + E_{I/O}$, where E_{CPU} is the energy consumption of the CPU, E_M is the energy consumption of the memory, and $E_{I/O}$ is the energy that is used for read and write operations. One very popular utilization based model was developed by Fan *et al.* [FWB07] and is defined as $P_{serv}(u) = (P_{max} - P_{idle}) * u + P_{idle}$, where P_{serv} is the server power consumption, P_{idle} is the power consumption of the server when it is idle, P_{max} is the maximum power consumption of the server, and u is the current utilization of the server. However, as for the accurate modeling of the impact of CPU frequency changes onto the server power consumption a power model that depends on the frequency is needed, these two models are not suitable to use in this thesis. A model, which depends on the CPU frequency of the server, was introduced by Elnozahy *et al.* [EKR02]. The model is defined as $P_{serv}(f) = Af^3 + P_{idle}$, where A is a server specific constant that includes the capacitance and the activity of the server gates and f is the CPU frequency of the server. Thus, A is dependent on the physical architecture of the utilized server. A slightly different version of this model was validated by Shoukourian *et al.* [SWA+15] on benchmark application data that were obtained from the SuperMUC system. The version of Shoukourian *et al.* is defined as follows:

$$P_{serv}(f) = k_1 f^3 + k_2 \tag{7.1}$$

Where k_1 and k_2 are application and server specific fitting parameters [SWA+15]. As this model was validated on data from the SuperMUC, it is used to model the server power consumption in the adjusted version of DCSim.

The structure of the model in Equation 7.1 is similar to a linear regression model that uses f^3 as its only variable. Thus, to fit the model to the available data trace, any tool that is able to perform linear regression fits can be used. In this thesis, the WEKA data mining framework [HFH+09] is utilized. The WEKA framework is an open source java-based data mining framework that was developed at the University of Waikato, New Zealand [HFH+09]. The fit of Equation 7.1 to the APC/Node values of the entire available workload data trace results in a rather poor fit that, when correlated with the original data, has a correlation (cf. Equation 7.2) of 0.3311. This is, because the workload trace contains many different application types and k_1 and k_2 are application dependent. Unfortunately, the data that are necessary to determine the application type of each job in the workload trace, are not available. As a solution, in this thesis, the data were clustered into 30 pseudo job classes and a separate fit of Equation 7.1 is created for each of these classes. The exact procedure of this solution is described in the remainder of this section.

In this thesis, the term correlation refers to the Pearson correlation coefficient between two variables. The Pearson correlation coefficient can be used to measure the linear correlation between two variables x and y and is defined as follows:

$$\rho_{x,y} = \frac{cov(x,y)}{\sigma_x \sigma_y} \tag{7.2}$$

Where $cov(x, y)$ is the covariance between variables x and y, σ_x is the mean of variable x, and σ_y is the mean of vairable y.

To create the job classes, the entire workload trace was clustered on the basis of the EtS, APC, and APC/Node values of each record. Thus, the records that end up in one cluster have similar power consumption characteristics, which is beneficial to jointly model the APC/Node values of these records, but they cannot be considered to be actually records from the same application type. The records were clustered by the k-Means implementation of the WEKA framework. The k-Means algorithm is a partitional clustering algorithm which tries to partition the input data into k spherical clusters. Thereby, it tries to minimize the distances between each datapoint and its assigned cluster centroid [Jai10]. For more information on the k-Means algorithm, the reader is referred to [Jai10]. As LoadLeveler assigns the execution frequency according to the application type of a job, the jobs which are from the same application type are always executed on the same CPU frequency [ABB+14]. Therefore, as in the majority of the 30 clusters only one frequency value exists, it is impossible to fit the power consumption model solely on the records from the workload trace.

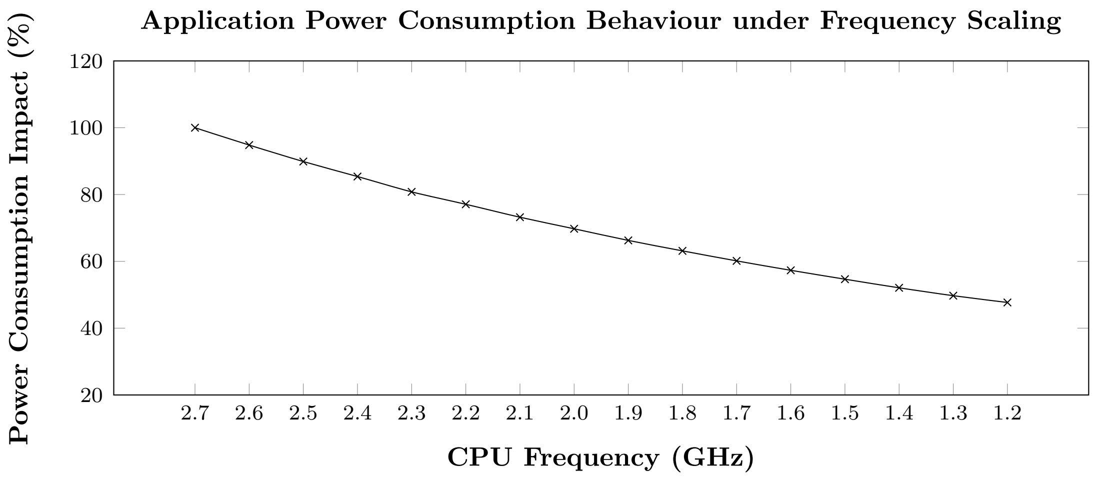


Figure 7.1.: Average application power consumption behavior under frequency scaling (data from [ABB+14]).

Thus, before a fit of Equation 7.1 is created for a job class, all job records within the class are scaled according to the average application power consumption behavior under frequency scaling, which was analysed on the SuperMUC in 2014 by Auweter *et al.* [ABB+14]. The observed average power consumption behavior of the analyzed application types is illustrated in Figure 7.1, where 2.7GHz were used as nominal frequency and an application's power consumption was normalized to be 100% at a frequency of 2.7GHz. After the scaling procedure, the dataset that is used to create the fit for a job class contains 16 records for each original record (one for each frequency that is illustrated in Figure 7.1). This ensures that the resulting model captures the application power consumption behavior under frequency scaling in a realistic way. The correlation of the original APC/Node values and the predicted APC/Node values from the 30 fits of Equation 7.1, a correlation of 0.98, which is a significant improvement compared to the correlation of a single fit to the complete workload trace, is obtained. However, the error that this approach introduces through the use of the average scaling behaviour for all jobs cannot be quantified as this is not possible with the available data. The described models are only used for servers that currently execute workload, for all idle servers a static value of 49W, which was suggested by the operators of the SuperMUC as a reasonable value, is used as the idle power consumption.

7.2. Batch Job Runtime Model

In order to model the impact of the application of DVFS on the impact of the runtime of jobs that are executed on the afftected servers correctly, either a model which models the runtime of a job in general or a model that only models the impact of a frequency change onto a previously given job runtime is needed. Shoukourian *et al.* [SWA+15] introduced a model of the first category that depends on the utilized nodes and the execution frequency of a job. It is defined as $TtS(n, f) = \frac{t_1}{f} + \frac{t_2}{n} + \frac{t_3}{nf} + t_4 n + t_5 \frac{n}{f} + t_6$, where TtS is a job's time to solution, f is the execution frequency, n is the amount of nodes the job utilizes, and $t_1 - t_6$ are constant fitting parameters that depend on the application type of the job [SWA+15]. However, as the jobs within the workload trace cannot be assigned to an application type and this model requires six application dependent fitting parameters, it is very difficult to fit one or several models of this structure to the available workload trace. Nevertheless, as all workload traces, which are utilized in this thesis, characterize the size of jobs in terms of a measured runtime for a specific execution configuration (execution frequency and amount of utilized nodes), it is also feasible to use models of the second category in this thesis. The model that is used for this thesis and thus, is also implemented in the adjusted version of DCSim is based on the fact that it is very easy to calculate the impact of a frequency change onto the runtime of a pure computation bound job that needs no main memory access. For such a job the impact on the runtime can be calculated as $t_n = (f_0/f_n)t_0$, where t_n is the job's runtime at frequency f_n and t_0 is the job's runtime at frequency f_0 [RLSdS11]. Unfortunately, it is very unlikely that real world jobs are purely CPU bound, which is also not the case for the typical workload of the SuperMUC [ABB$^+$14]. As the memory access times are not sensitive to the CPU frequency, the impact of a frequency adjustment onto the runtime of a job changes with the degree of memory boundedness of the job [RLSdS11]. Hsu and Kremer [HK03] introduced a model that tries to incorporate the memory boundedness of a job. Within this thesis a slightly adjusted version of this model, which was introduced by Etinski *et al.* [ECLV12], is used. The slightly adjusted version is defined as follows:

$$\frac{T(f)}{T(f_{max})} = \beta(\frac{f_{max}}{f} - 1) + 1 \tag{7.3}$$

Where $T(f)$ is the job's runtime at frequency f, $T(f_{max})$ is the job's runtime at a nominal frequency f_{max}, and β is a fitting parameter that depends on the degree of memory boundedness of a job. This model is derived under the assumption that the computation time T_{CPU} changes inversely proportional to the frequency, whereas the memory access time T_{MEM} remains constant [ECLV12]. Thus, β can also be expressed as [ECLV12]:

$$\beta = \frac{T_{CPU}(f_{max})}{T_{CPU}(f_{max}) + T_{MEM}} \tag{7.4}$$

Therefore, a value of $\beta = 0$ indicates a purely memory bound application and a value of $\beta = 1$ indicates a purely CPU bound application [ECLV12].

In 2014, Auweter *et al.* [ABB$^+$14] analyzed this relation for a the same range of application types of the SuperMUC for which they also analysed the power consumption behviour (cf. Figure 7.1). The results are illustrated in Figure 7.2. 2.7GHz were used as the nominal frequency for the analysis. When one fits the model from Equation 7.3 to the shown runtime behavior for each (f_{max}, f) pair, although according to Equation 7.4 the β value should be constant for all frequency pairs, fifteen different β values are obtained (one for each (f_{max}, f)

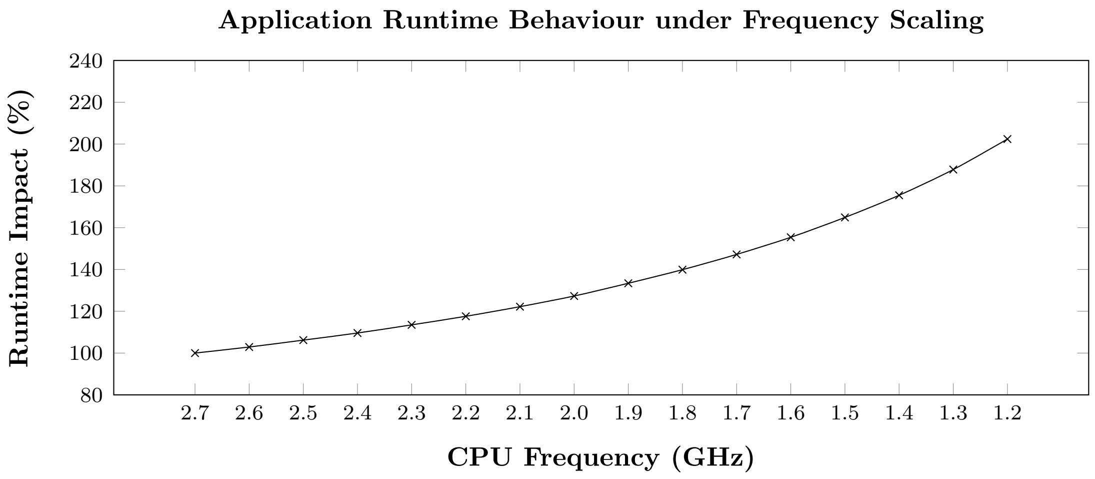


Figure 7.2.: Average application runtime behavior under frequency scaling (data from [ABB+14]).

tuple). However, as the maximum deviation of the β values from 0.79, which is the center value of the range of the fifteen values, is 4.5% and, according to Etinski *et al.* [ECLV12], it is reasonable to use a single value for β if the maximum deviation of the other values is such small, the value of $\beta = 0.79$ can be used for all frequency pairs. In the adjusted version of DCSim, the runtime impact is calculated on job level.

7.3. IT Power and Cooling Power Models

The model, which is described in Section 7.1, only captures the server power consumption of the DC, but not the power consumption of the PDUs, storage equipment, lighting, and network equipment of the DC. Therefore, to estimate the IT power, which incorporates all the server power consumption and the power consumption of the additional IT related equipment, on the basis of the modeled server power consumption, the following relation was analyzed:

$$x = \frac{P_{IT} - P_{serv}}{P_{serv}} \tag{7.5}$$

Where P_{IT} is the IT power consumption and P_{serv} is the server power consumption. As the available IT power consumption trace is a hourly based time series trace, but the available job data are on job basis, to perform this analysis, the cleaned job data trace was aggregated to hourly time series data by the use of an existing java code that basically checks for every second in 2014 how much jobs (amount of running jobs) and nodes (amount of active nodes) were active and how much power (job power) they drew and takes the average of these values for all seconds in one hour as the aggregated value for the corresponding hour. The time series data for P_{serv} were obtained under the assumption that all nodes that are currently not active consume 49W. When this is the case, P_{serv} equals the sum out of the job power and the power consumption of all idle servers.

x indicates the fraction of the server power consumption that is equal to the difference between IT power consumption and server power consumption. This fraction can be used to calculate an estimated IT power consumption from the server power consumption. The median value of x is 0.4 for the available data traces, which means that roughly 71% of the IT power is consumed by the servers and the remaining 29% are consumed by the rest of the IT equipment. For the estimated IT power on the basis of the original server power trace and a x value of 0.4, a mean absolute error (MAE) of 139.99, a mean absolute percentage error (MAPE) of 8.75%, and a correlation of 0.85 is obtained. Nevertheless, as the purpose of the adjusted simulation framework is not to exactly rebuild the behaviour of the SuperMUC, but of an exemplary large scale DC, the accuracy of this estimation is sufficient. The MAE and MAPE measures can be used to quantify the prediction accuracy of a prediction model. The two measures are calculated as follows:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n} \tag{7.6}$$

$$MAPE = \frac{\sum_{i=1}^{n} |\frac{y_i - \hat{y}_i}{y_i}|}{n} \tag{7.7}$$

Where y_i are measured values and $\hat{y}_i$ are predicted values.

Shoukourian *et al.* [SWLB17] created a neural network model that is capable of predicting the COP of the chiller-less cooling infrastructure of the complete building that houses the SuperMUC system. The COP is defined as $COP = \frac{Q_{CoolingCircuits}}{P_{CoolingCircuits}}$, where $Q_{CoolingCircuits}$ is the amount cold that is generated by the cooling infrastructure and $P_{CoolingCircuits}$ is the amount of power that is consumed by the cooling infrastructure [SWLB17]. Therefore, the COP could also be used to determine the power consumption of the cooling infrastructure. Shoukourian *et al.* used the amount of cold generated by the cooling infrastructure, the amount of power consumed by the fans of the cooling towers, the amount of active cooling towers, the wet bulb temperature, the inlet water temperature of the cooling infrastructure, and the outlet water temperature of the cooling infrastructure as an input for their model [SWLB17]. As, out of these values, only the wet bulb temperature is partially available for this thesis, it is infeasible to reproduce this model in order to use it as power consumption model for the HVAC infrastructure. Furthermore, as the model is a neural network, which tend to work well for non-linear problems, it is likely that the relation between the input parameters and the power consumption of the cooling infrastructure is not linear. However, as the considered DC calculated the values of the PUE data trace by the use of Equation 4.1, the PUE values can be used to estimate the cooling power of the DC. The cooling power can be calculated through the following equation:

$$CoolingPower = (PUE * ITPower) - ITPower \tag{7.8}$$

7.4. SLA Model

As the SLA model which was originally implemented in DCSim is only suited for interactive workloads, it had to be adjusted to fit Requirement **R4** (cf. Section 6.1). The adjusted SLA model is based on a definition of Garg *et al.* [GTGB14]. Garg *et al.* defined the QoS (Quality of Service) requirements of a batch job in terms of a deadline and the amount of CPU cycles that are allocated to a batch job [GTGB14]. When the job finishes before the deadline (if sufficiently much CPU cycles were allocated to the job before the deadline), there are no penalty costs. However, when the job finishes after the deadline, the penalty can be calculated according to the equation $Q(batch) = y * db$, where y is a fixed penalty rate and db is the delay [GTGB14]. For this thesis, this model is slightly changed to also incorporate the amount of nodes that are utilized by a batch job. The changed model, which is implemented in the adjusted version of DCSim, is defined as $Q(batch) = uP * rD$, where uP is the usage price (in €) of an entire job and rD is a relative delay. The usage price of an entire job is calculated on the basis of the amount of nodes that a job utilizes and the runtime that was originally specified in the workload trace. In the adjusted implementation a default usage price of 0.36€ per hour per node, which is charged by a HPC in Stuttgart [(HL16] for a similar type of node as the ones that the SuperMUC contains, is used. The relative delay is calculated as $rD = (ActualFinishTime - SLADeadline)/defaultRuntime$, where $defaultRuntime$ is the runtime of a job that was originally specified in the workload trace and $pD = 0$ when the job is not delayed.

As the job data trace does not provide deadline values for the batch jobs, the deadlines have to be artificially generated. This is done according to a procedure that is also proposed by Garg *et al.* [GTGB14]. They assume that a typical deadline of a batch job can be calculated according to the following equation [GTGB14]:

$$deadline = jobStart + runtime + (runtime * uniform(0, 2)) \tag{7.9}$$

Where $deadline$ is the time of the batch job deadline, $jobStart$ is the point in time where the job starts, $runtime$ is the time that the job needs under optimal execution conditions, and $uniform(0, 2)$ is a sample from a uniform distribution in the interval between 0 and 2. For the simulation in this thesis, the runtime of the jobs is calculated from the start and end times which are specified in the job data trace. To ensure that the simulation runs are deterministic, a SLA deadline is generate once for the entire cleaned job data trace and added to the trace.

8. Evaluation

This chapter describes the objective, approach and results of the evaluation that was carried out during the thesis. Therefore this chapter is structured into three sections. Section 8.1 describes the objectives and approaches that were used for the evaluation. In Section 8.2 a validation of the simulation framework that was developed during the thesis is provided. Finally, Section 8.3 introduces the setting and describes and discusses the results of the evaluation that was carried out to analyse the potential of a large scale DC to participate in a DR market.

8.1. Objective and Approach

The first objective of the evaluation is to validate that the adjusted simulation framework is able to simulate the behaviour of a large scale real world DC in a realistic way. To do so, the simulation framework is used to simulate the entire year of 2014, on the basis of the cleaned job data trace (cf. Subsection 4.2.1) without any DR requests. Afterwards, to validate that the simulator produces realistic values, the simulated records are compared to the originally available data traces. As the original data traces for total facility power, cooling power, IT power, job power, amount of active nodes, and amount of running jobs are on hourly basis, but the monitoring records of the simulation are on a secondly basis, they were also aggregated to form hourly time series. The original time series traces for job power, amount of active nodes, and amount of running jobs time series were obtained from the cleaned job data trace through the procedure that is decsribed in Section 7.3. However, during this aggregation, it is not checked whether the amount of available nodes is exceeded by the specified schedule at any point of time. The monitoring records of the simulation framework were aggregated by the use of the `SuperMUCEvaluation` class.

The second objective of the evaluation is to evaluate the potential of a large scale DC to participate within a real world DR market. To reach this objective, the participation within such a market is simulated for one week of 2014, on the basis of the cleaned job data trace. Subsequently, the costs that emerged during the simulated participation in a DR market are compared to the costs of the validation run, which is taken as a baseline for the evaluation.

The configuration parameters, which are described in the remainder of this section, are used, if not explicitly stated otherwise, for the validation run and all DR market participation runs. As the precision of the times in the provided workload trace is on seconds basis, a simulation step length of one real time second is used. The scheduling interval length is set to one simulation step. This is necessary, because the minimum time difference between the submission time and start time is 0 seconds in the provided data trace. Therefore, for larger scheduling intervals it cannot be guaranteed that the original schedule of the SuperMUC is rebuilt. As it is the purpose to rebuild the schedule of the SuperMUC, the *superMUC-Mode* is used. For the energy price, the static average industrial energy price of 0.1532€/kWh [dEA] from 2014 is used and for the usage price of one compute node hour the price of 0.16€/NodeHour, which a HPC in Stuttgart [(HL16] charges for a similar node type, is used. For the PUE values, the provided PUE trace (cf. Subsection 4.2.2) is utilized. The simulation is started at the 01.01.2014 00:00:00. For the validation run, the end time is set to the 01.01.2015 00:00:00 and for the runs for the evaluation of the participation in a DR market it is set to the 20.08.2014 (to ensure that all possibly caused SLA costs are captured).

8.2. Validation of the Simulation Framework

This section provides and discusses the results of the validation run of the simulation framework.

Table 8.1 shows the statistics of the comparison between the original data traces and the corresponding simulated data traces. Besides already introduced statistics, the table includes the coefficient of determination (R^2), which is used to quantify how well a model replicates the actually observed values. For simple bivariate linear regressions, the R^2 measure equals the square of the correlation coefficient between the observed and modeled values. The R^2 measure is defined as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y_i})^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{8.1}$$

Where y_i are actually observed values, $\hat{y_i}$ are predicted values, and $\bar{y}$ is the mean of the actually observed values.

The high correlation (0.985) and R^2 (0.97) values between the original and the simulated job data traces indicate that the simulation framework is able to reproduce the job power consumption of the SuperMUC very accurately on the basis of the pseudo job classes. Another indicator for this is the rather low MAPE of 4.37%. The accuracy of the simulated IT power consumption, total facility power consumption and cooling power consumption is not as accurate as the simulated job power consumption. This is indicated by the lower correlation (0.812, 0.808, and 0.896), lower R^2 (0.659, 0.654, and 0.803), and higher MAPE (10.11%, 10.11%, and 10.11%) values. The reason for this is that the derived data traces of total facility power and cooling power are both based on the IT power consumption of the DC. The statistics of the simulated IT power consumption are slightly worse than stated in Section 7.3, because the simulated job power consumption does not exactly reproduce the original job power consumption trace. Interesting to note is that the correlation of the cooling power consumption traces is much higher than the correlation of the IT power consumption and total facility power consumption traces. As both, the cooling power consumption and the total facility power consumption traces are calculated based on the IT power consumption, one would expect that they have a correlation that is similar to the correlation of the IT power traces. This is also the reason for the similar MAPE values of the cooling power consumption, the total facility power consumption,

Table 8.1.: Statistics of the comparison between original data traces and simulated data traces.

	Correlation	**R^2**	**MAE**	**MAPE**
Job Power	0.985	0.97	51.2	4.37%
IT Power	0.812	0.659	165.764	10.11%
Cooling Power	0.896	0.803	21.508	10.11%
Total Facility Power	0.808	0.654	187.269	10.11%
Active Nodes	0.999	0.999	1.329	0.03%
Running Jobs	0.999	0.999	0.007	0.02%

and IT power consumption. However, the different correlations are caused by differences of the general size of the cooling power consumption and the total facility power consumption. As the cooling power consumption is only a small part of the total facility power consumption, the variation that is introduced by the dynamic PUE value has a much larger impact on the cooling power consumption than it has on the total facility power consumption. Nevertheless, as the MAPE for all three kinds of power consumption is rather small ($\approx 10\%$), the accuracy of the simulated power consumption traces is sufficient for the purpose of this thesis.

The statistics also show that the simulation framework is able to exactly reproduce the schedule which is specified by the start and end times within a provided workload trace. However, that the two correlation values for active nodes and running jobs are not exactly one indicates that there are small differences between the simulated traces and the original traces. These differences appear, because within the cleaned job data trace, there are some spots at which the originally specified schedule exceeds the amount of available nodes. This was not considered during the aggregation of the original time series, but is considered by the simulation framework during the simulation. Therefore, in the simulation, some jobs are not started at their originally specified start time, but slightly after this time.

8.3. Demand Response Potential of a Data Centre

This section introduces the setting that is used for the simulation of the participation of a large scale DC within a DR market, provides the results of this simulation, and discusses them.

8.3.1. Setting

To evaluate the potential of the successful participation of a large scale DC within a DR market, the participation of a simulated DC in the secondary reserve market in Germany is considered. The secondary power reserve is used by the grid operators to compensate possible power load variations that cause grid instabilities. An activation period in the secondary reserve market has a maximum length of 15 minutes and the reserve power providers have to reach their offered adjustment height no later than 5 minutes after they received the activation request. The secondary reserve market is auction based and in 2014 the auctions took place on a weekly basis. Per week their were four separate auctions, one for each possible combination of the possible provision times and directions [NEX]. The provision times are separated into main time (monday to friday, 8:00-20:00) and secondary time (holidays, sundays, saturdays, and monday to friday 0:00-08:00 and 20:00-24:00) [BW]. The possible reserve types are positive and negative reserves (cf. Subsection 6.3.5). Besides the specification of a combination of provision time and provided reserve type, a bid also includes the maximum amount of provided reserve power (in MW) a provided power compensation price (PPCP) (€/MW) and a provided energy compensation price (PECP) (€/MWh). The PPCP specifies the price that is paid by the grid operator to the power reserve provider, when the bid of the provider is accepted by the grid operator. In contrast, the PECP specifies the price that the grid operator pays to the provider for each MWh of reserve power that the provider actually provided during the corresponding week. To determine which bids are accepted, the grid operator sorts all bids according to their PPCP and afterwards, accept the ones with the lowest PPCPs until the sum of the offered reserve power fulfills the reserve power demand of the grid operator [NEX]. Thus, the best PPCP for one auction week, is the PPCP of the last accepted bid. The grid operators use a similar procedure on the basis of the PECPs to determine, which providers are actually activated for reserve power provision. Therefore, providers that offer rather low PECPs are activated more often than providers that offer very high PECPs [NEX].

In 2014 the minimum reserve power that a provider had to offer was 5MW, which is much larger than the maximum reserve power that the SuperMUC can offer. However, it is assumed that lower amounts of reserve power can be offered through an aggregator party that functions as one provider in the secondary reserve market, but actually aggregates the reserve power of several actual providers [Bun16]. For this evaluation, one the week (and thus one bidding period) which starts at the 03.03.2014 and ends at the 09.03.2014 is considered. To make the evaluation as realistic as possible, some actual PPCP and PECP values from 2014 are used to construct the artificial bid that the simulated DC has used for the auction. In addition to the prices, there are also historic data from 2014 available that state how much secondary reserve power was requested during each 15-minute interval. All these data are accessed via *regelleistung.net*[1], which is the internet platform on which the auctions for the different reserve power markets take place. This enables the possibility to determine whether the offer, which the simulated DC offered during a specific week in 2014, was activated or not during each 15-minute interval of the corresponding week. For the evaluation it is assumed that the simulated DC only participates in the auction for positive reserve power provision during the main time. During the considered week in 2014, the maximum accepted PPCP, for positive reserve power, was 382€/MW and the PECP of the provider that earned the most revenue in this week was 63.1€/MWh. The provider that offered this PECP was activated in 90 15-minute intervals during the considered week in 2014. Thus, the DR request trace that is used for the simulation of the participation in the secondary reserve market, also has 90 entries, where each entry specifies a DR event request at the same time at which the real world provider was activated. The artificial offer that combines these two prices, in theory, reaches the maximum possible revenue from the participation in the secondary reserve market. Therefore, they are used to calculate the revenue which the simulated DC achieves by the participation in the secondary reserve market. The adjustment height, adjustment length, and provision type values are equal for all 90 entries in the DR event trace. This is the case, as the DC has to offer the same product for the entire week. Thus, the provision type will be set to positive and the adjustment length will be set to 15 minutes for all entries. The maximum amount of provided reserve power (adjustment height) is set to several levels (0.1MW, 0.2MW, 0.3MW, 0.4MW, 0.5MW, 0.6MW, 0.7MW, and 0.8MW). For simplicity reasons, it is assumed that during each DR event, the total offered reserve power has to be provided. Furthermore, it is assumed that the DR reacts to the request in real time and thus, provides the total offered reserve power for the whole timespan of the DR event. Consequently, the reward, which is received for one DR event, can be calculated as $Reward = PECP * OfferedReservePower * 0.25$.

[1]`https://www.regelleistung.net/ext/tender/`

8.3.2. Results and Discussion

For the runs where 0.8MW and 0.7MW were offered, it was obtained that several DR requests during the simulated week were not handled successfully, because the required power adjustment bound was exceeded. Thus, the results of these two simulation runs are excluded from the remainder of this section.

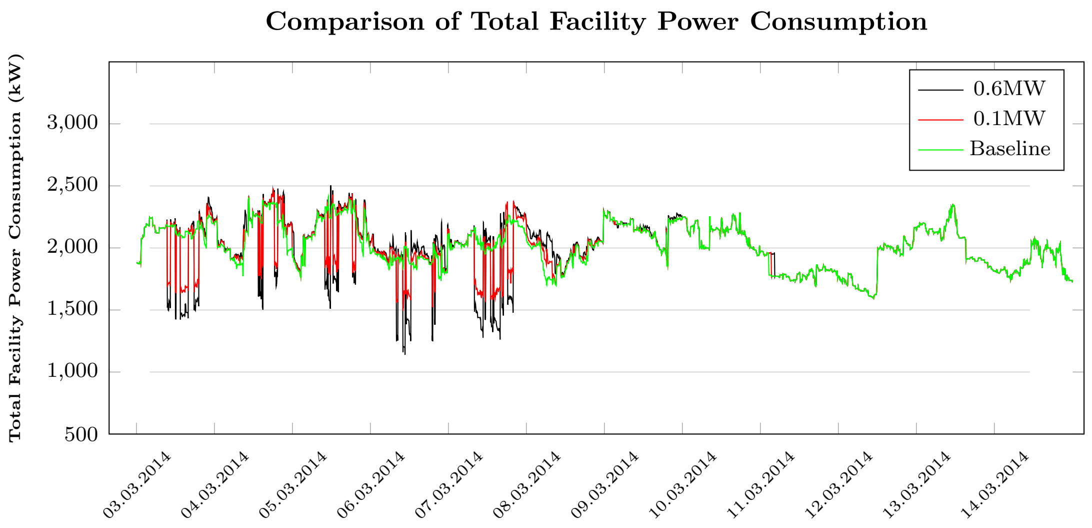


Figure 8.1.: Comparison of three total facility power consumption traces (Baseline, 0.6MW, and 0.1MW) in the interval from the 03.03.2014 00:00 to the 14.03.2014 23:59.

Figure 8.1 shows the total facility power consumption traces, which were aggregated to time series data on the basis of 5 minute intervals, from the baseline run (green), the 0.1MW provision run (red), and the 0.6MW provision run (black) in the interval from the 03.03.2014 00:00 to the 14.03.2014 23:59. The DR events can be clearly identified in the figure, as the total facility power consumption of the provision runs clearly deviates downwards compared to the baseline run. Thereby, as expected, because of a lower adjustment boundary, the total facility power consumption of the 0.6MW provision run decreases further than the one of the 0.1MW provision run. A similar effect can be seen on the upward deviations from the reserve power provision runs in comparison to the baseline run. That is also expected, because due to the fact that during the 0.6MW provision run the simulated DC had, in comparison to the one in the 0.1MW provision run, to provide more reserve power, which leads to more delayed workload that is executed after the end of the DR events. One can also see that the total facility power consumption of the reserve provision runs is often higher than the baseline power consumption directly after DR events. This is reasonable, as at this points in time possibly shifted jobs are started. Also interesting to note is that although, in theory, the simulated DC in the 0.1MW provision run is requested to provide 0.5MW less reserve power than in the 0.6MW provision run, it is observed that the difference between the two power consumption profiles of the two runs is almost always much smaller than 0.5MW. This is, because the simulated DC tries to pick the energy optimal run configuration for each job that satisfies the requested power consumption bound, but the default job run configuration of the real world SuperMUC is not energy optimal as the optimal configuration would decrease the productivity of the SuperMUC significantly [ABB+14]. Although the week that was considered for the participation in the secondary reserve market ends at the 09.03.2014, one can see that the last executed DR event was at the evening of the 07.03.2014. This is, because the 8th and 9th of March of 2014 are on a week-end and thus belong to the secondary time market. It is also noticed that the total facility power consumption of the reserve provision runs needs some time after the end of the last DR event to adjust back to the total facility power consumption trace of the baseline run. During this time, possibly shifted and delayed workload from the DR events was executed. The point at which all three traces were equal again was on the 11.03.2014, where a drop of the two reserve power provision runs can be seen in the figure. The drop of the 0.6MW run is slightly after the drop of the 0.1MW run, which is most likely because during the 0.6MW provision run more reserve power was provided.

Figure 8.2 illustrates the amount of active nodes traces, which were also transformed to time series data on the basis of 5 minute intervals, from the baseline run (green), 0.1MW provision run (red), and the 0.6MW prvision run (black) in the interval from the 03.03.2014 00:00 to the 14.03.2014 23:59. One can see that the amount of active nodes traces of the two provision runs do not deviate from the baseline trace as much as it was the case for the total facility power consumption. It is also not possible to identify the DR events from the active amount of nodes traces. This suggests, that for the provision of the reserve power during the provision runs not much workload was actually shifted, but the vast majority of the reserve power was provided through the application of DVFS. A possible

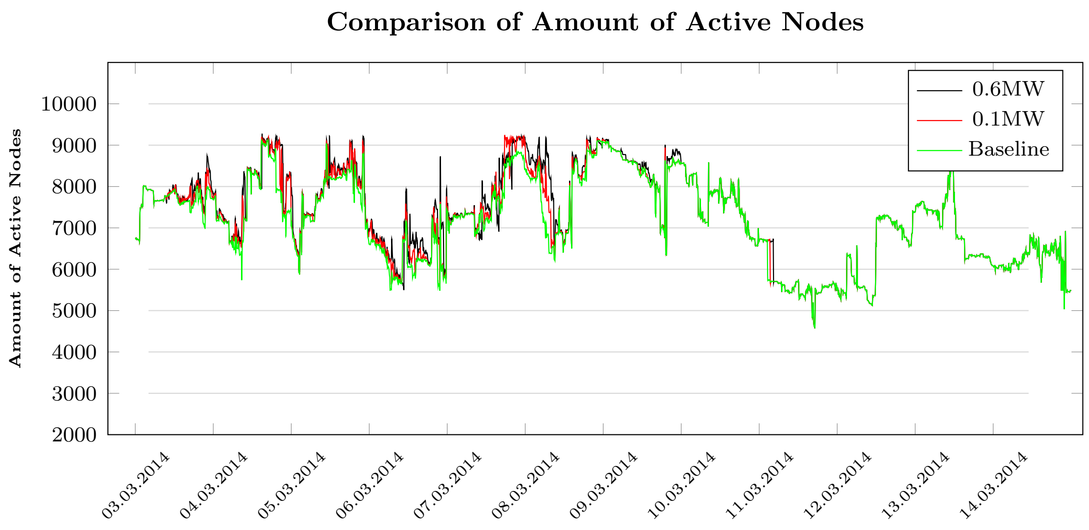


Figure 8.2.: Comparison of three amount of active node traces (Baseline, 0.6MW, and 0.1MW) in the interval from the 03.03.2014 00:00 to the 14.03.2014 23:59.

reason for this is that it is assumed that jobs cannot be paused, which means that all jobs that run at the beginning of a DR event cannot be shifted. The upward deviations of the provision run traces in comparison to the baseline trace are most probably caused by the jobs that got actually shifted and are started after the ends of the DR events.

Table 8.2.: Statistics of the used shifting fraction configurations.

	0.6MW	**0.5MW**	**0.4MW**	**0.3MW**	**0.2MW**	**0.1MW**
Average	0.3	0.09	0.08	0	0	0
Median	0	0	0	0	0	0
Max	1	0.89	0.98	0.38	0	0.01
Min	0	0	0	0	0	0
Std. Dev.	0.39	0.21	0.2	0.04	0	0

Table 8.2 shows several statistics for the shifting fraction configurations that were used during the reserve provision simulation runs to react to the DR events. It can be seen that the shifting fraction that is used on average decreases with the amount of reserve power that is provided, which is reasonable as during the runs with lower reserve provision offers less adjustments to the power profile are necessary. One can also see that the maximum shifting fractions are quite high, although in Figure 8.2 it is shown that not much workload is actually shifted.

However, these two findings do not contradict each other, because, as the current implementation does not include a predictor for upcoming workload, the shifting fractions in the configurations only state the shifted fractions of the workload that was **actually shiftable** at the start of a DR event window. Due to the assumption that jobs cannot be paused, this includes only the jobs that were already submitted to the scheduler, but have not started at the beginning of a DR event. As the average difference between the submission time and the start time in the utilized workload trace is very low, the number of jobs that the scheduler already knows at the beginning of a DR event is likely to be very small.

Table 8.3.: Statistics of the used scaling frequency configurations.

	0.6MW	**0.5MW**	**0.4MW**	**0.3MW**	**0.2MW**	**0.1MW**
Average	1.71	1.86	1.97	1.998	2	2
Median	1.7	1.9	2	2	2	2
Max	1.9	2	2	2	2	2
Min	1.3	1.6	1.8	1.9	2	2
Std. Dev.	0.13	0.09	0.05	0.01	0	0

Table 8.3 states several statistics of the scaling frequencies that were used during the reserve power provision runs for the reaction to the DR events. It is observed that the scaling frequency, which is used on average, increases when the amount of offered reserve power decreases. This is reasonable, as the lower frequencies are needed to provide larger amounts of reserve power. However, one can see that the median frequency that was used during the 0.4MW run, 0.3MW run, 0.2MW run, and 0.1MW run is 2GHz. This suggests that the optimal frequency, which is obtained through the currently implemented optimization procedure (cf. Subsection 6.3.6), is lower than the minimum frequency that would be necessary to decrease the power consumption below the required power consumption bound. That means, as during all runs except the 0.6MW provision run no SLA costs occurred (see Table 8.4), that the energy optimal execution frequency of a job is 2GHz in the current implementation of the simulation framework. This value slightly differs from the value of 1.8 GHz which was determined for the SuperMUC by Auweter *et al.* [ABB$^+$14]. Most likely, this slight difference is caused by the use of the average scaling behaviour for both, the runtime and the power consumption of the jobs.

Tables 8.2 and 8.3 show that no default optimal combination of power demand flexibility provision techniques can be defined for the simulated setting, but that the optimal combination depends on the amount of provided reserve power and the workload structure at the point in time when the DR event is received. However, it is observed that over all provision runs, the combination of 0% workload shifting and 2.0GHz scaling frequency was very commonly used.

Table 8.4.: Comparison of costs between baseline run and provision runs.

	0.6MW	**0.5MW**	**0.4MW**	**0.3MW**	**0.2MW**	**0.1MW**	**BL**
E1	88089.85	88127.15	88178.88	88211.84	88212.85	88212.34	89004.09
E2	13799.98	13805.82	13813.92	13819.09	13819.25	13819.16	13943.2
SLA	2787.8	0	0	0	0	0	0
PEC	851.9	709.9	567.9	425.9	283.9	141.98	0
PPC	229.2	191	152.8	114.6	76.4	38.2	0
C1	89796.55	**87226.25**	87458.18	87671.34	87852.55	88032.16	89004.09
C2	15506.68	**12904.92**	13093.22	13278.59	13458.95	13638.98	13943.2

Table 8.4 shows two different energy cost values E1 (static energy price) and E2 (dynamic energy price), the SLA costs (SLA), the received power provision compensation (PPC), and the total sum of received energy provision compensation (PEC) of all reserve provision runs and the baseline run. Furthermore, it shows two total cost sums C1 and C2, where C1 is calculated as $C1 = E1 + SLA - PEC - PPC$ and C2 is calculated as $C2 = E2 + SLA - PEC - PPC$. All values that are displayed in Table 8.4 are calculated for the interval from the 03.03.2014 00:00 to the 14.03.2014 23:59. This is done to capture the differences of the total facility power consumptions of the reserve provision runs and the baseline run after the 09.03.2014. Although the simulation runs were carried out by the use of a static average energy price, as the current shifting procedure does not consider a possibly changing energy price, the resulting power profile would not change when a dynamic energy price would be used. Thus, the energy costs E2 can be calculated on the basis of the power profile from the simulation runs that were executed with a static energy price. E2 is calculated on the basis of the day ahead spot market auction price data of the EPEX[2] in 2014.

The dynamic spot market price for the interval from the 03.03.2014 00:00 to the 14.03.2014 23:59 is shown in Figure 8.3. One can see that the prices are much lower than the static average price of 0.1532€/kWh.

[2]`http://www.epexspot.com/de/`

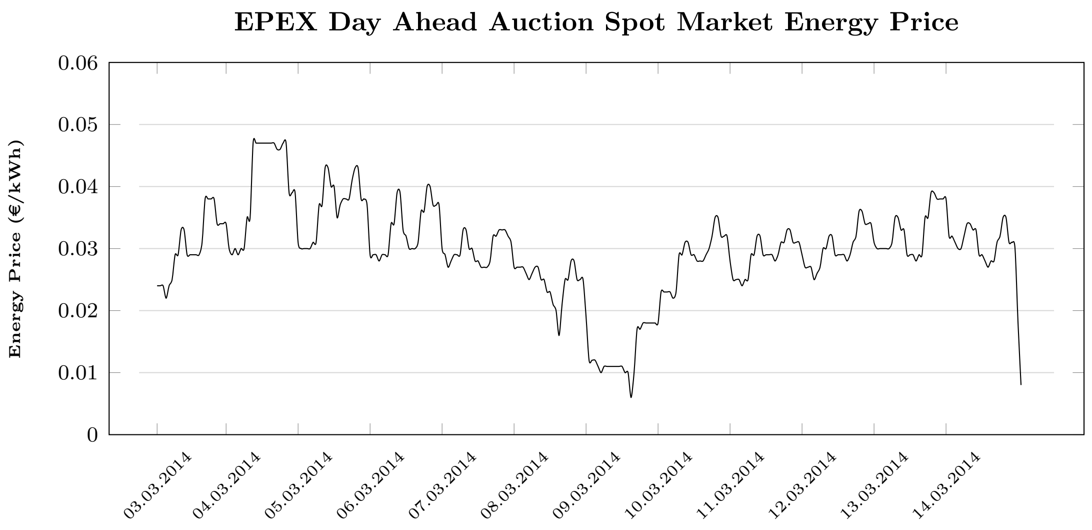


Figure 8.3.: EPEX day ahead auction spot market price in the interval from the 03.03.2014 until the 15.03.2014.

Interesting to note is that both energy cost values E1 and E2, are slightly higher for the provision runs in which smaller amounts of reserve power were provided. In the case of the static energy price based costs E1, the energy costs only change when the execution frequency is adjusted. That means that these small differences occur, because during the runs with higher amounts of provided reserve power, lower scaling frequencies were used. This suggests that the optimization of the energy consumption costs on job level does not necessarily optimize the energy consumption costs on the total DC level, because if this would be the case, all power provision runs without SLA costs should have equal energy costs. That the energy costs of all provision runs are smaller than the energy costs of the baseline run shows again that the SuperMUC does not use an energy optimal default run configuration. As the dynamic energy prices which are used for the calculation of E2 are much smaller than the utilized static energy price, this effect has less impact onto the E2 prices. Among the provision runs that successfully responded to all requested DR events, the only run for which SLA costs occurred is the run in which 0.6MW of reserve power were provided. These costs are higher than the rewards that are received for the participation in the DR market, which makes it economically infeasible for the simulated DC to provide 0.6MW of reserve power. The provision run with the lowest total costs is the one in which 0.5MW of reserve power were provided. This is the case for both cost values C1 and C2. Thus, for the simulated DC the optimal amount, out of the set of tested amounts, of provided reserve power would be 0.5MW. When the simulated DC provides 0.5MW, the participation within the DR market leads, in comparison to the run where it did not participated in the DR market, to savings of 1777.9€ for C1 and 1038.28€ for C2. Thus, it is shown that the DC has the potential to successfully participate in the secondary reserve market in Germany. In both cases, the energy costs that are saved by the reduced energy consumption equal about 1.6% of the baseline costs (53283.61 for E1 and 8347.3 for E2) in the actual participation week (03.03.2014 00:00 until 09.03.2014 23:59). Basically, it is expected that this percentage is higher for the dynamic price based energy costs, because the dynamic energy prices should go up when the energy demand is higher than the energy production and go down in the opposite case. Thus, DR events are likely to happen in times where the dynamic price is high. However, a possible reason for this is that the considered dynamic price is from a day ahead auction and thus the volatilities in the prices rely on the predicted amounts of power production and demand. In contrast, the secondary reserve is used by the grid operators to compensate for unpredictable volatilities in the power grid [BW]. Thus, the events in the secondary reserve market might not directly match the peaks in the day ahead spot market prices. The fraction of the savings which are reached through the reduced energy consumption during the response to the DR events is around 49% when the static energy price is used and about 13% when the dynamic energy price is used. The remaining 51% respectively 87% of savings come from the rewards that are earned for the participation within the secondary reserve market. In the real world, the SuperMUC does not use a more energy efficient run configuration, because the impact on the productivity would be too high [ABB+14]. However, in the considered week, the power consumption had to be adjusted for a total of 22.5 hours, which is about 13% of the complete time of the week. Therefore, the impact onto the productivity is much lower when a more energy efficient configuration is only used for 13% of the time and the savings that can be reached during the adjustment times increase, dependent on the energy acquisition method of the DC, significantly by the participation in the DR market. This might make it interesting for the operators of real world large scale HPC systems to consider the participation in a DR market.

9. Conclusion

This chapter concludes this thesis by a summary of the work that was done, a discussion of the limitations of the thesis, and some suggestions for future work. Therefore, the chapter is structured into three sections. Section 9.1 summarizes the contribution and findings of this thesis. Section 9.2 briefly discusses the limitations of this thesis. Finally, Section 9.3 suggests some ideas for follow up work for this thesis.

9.1. Summary

The main objective of this thesis was to develop a simulation framework that provides the possibility to simulate the impact of the provision of power demand flexibility onto the energy- and power use, as well as the energy- and SLA costs of a large scale DC. The two power demand flexibility provision techniques that were considered for the implementation of the simulation framework are Workload Shifting and Dynamic Voltage and Frequency Scaling. For the development of the simulation framework, the DCSim simulation framework, which was originally developed for the All4Green project[1], was taken as a basis. The main adjustments that were implemented during this thesis, are the introduction of the possibility to simulate the execution of scientific batch workload and the creation of models and mechanisms that enable the possibility to simulate the application of the considered power demand flexibility provision techniques within a DC. The developed simulation framework was designed in a way that makes it very easy to exchange the used models and strategies. This makes it possible to use the simulation framework to simulate and evaluate various different real world scenarios. For the validation and evaluation runs in this thesis, the data traces from the SuperMUC, which is a real world large scale HPC in Germany, were utilized. During the validation, it was validated that the simulation framework is

[1]`http://all4green-project.gfi.es/`

able to reproduce the power consumption profile of the SuperMUC on the basis of its workload trace. In addition, it was shown that the developed simulation framework can be used to simulate the participation within the secondary reserve market in Germany.

The second objective of this was to evaluate whether a large scale DC can increase its revenue through the participation in a DR market. The real world DR market that was considered for this evaluation is the secondary positive reserve power market in Germany. It was shown that the DC, which is simulated on the basis of the SuperMUC traces, is able to increase its revenues by the participation in the secondary positive reserve power market. For the evaluation scenario, the simulated DC gained up to 12% of the total energy costs of a DC in the period of participation. This shows that it can be beneficial for large scale DCs to participate in DR markets and that this option should be considered by the operators of real world DCs. During the evaluation it was observed that it is not possible to define a combination of power demand flexibility provision strategies that is optimal in all situations. In the evaluated scenario, the structure of the optimal combination depended on the amount of offered reserve power and the workload structure at the point in time at which a DR event is received. However, it was found that for the majority of the evaluation runs, the combination of 0% workload shift and an execution frequency of 2GHz works well on average.

9.2. Limitations of the Thesis

A major limiting factor for this thesis was that the data were not sufficient to validate that the simulation framework models the impact of power demand flexibility provision onto the energy- and power use, as well as the energy- and SLA costs of a large scale DC realistically. Due to the limited time span of this thesis it was not possible to evaluate the performance of more than one scheduling strategy as well as the participation in more than one DR market.

9.3. Future Work

In the future, when sufficient data is available, it has to be validated that the simulation framework models the impact of power demand flexibility provision onto the energy- and power use, as well as the energy- and SLA costs of a large scale DC realistically. Furthermore, it would be interesting to evaluate the performance of different scheduling strategies and/or modeling approaches. Another interesting topic for future work is to evaluate the benefits that a DC which uses a virtualized execution environment can gain by the participation in a DR market. As in that case jobs actually can be paused, it is expected that the power demand flexibility that such a DC can provide is much larger than that of a DC which uses a non-virtualized execution environment.

Bibliography

[ABB$^+$14] Axel Auweter, Arndt Bode, Matthias Brehm, Luigi Brochard, Nicolay Hammer, Herbert Huber, Raj Panda, Francois Thomas, and Torsten Wilde. A case study of energy aware scheduling on supermuc. In *International Supercomputing conference*, pages 394–409. Springer, 2014.

[Age18] International Energy Agency. Monthly electricity statistics. online publication, January 2018. Url: `https://www.iea.org/media/statistics/surveys/electricity/mes.pdf`, Accessed: 01.02.2018.

[AR14] Baris Aksanli and Tajana Rosing. Providing regulation services and managing data center peak power budgets. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 143. European Design and Automation Association, 2014.

[BCH13] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 8(3):1–154, 2013.

[BEN04] Michel Berkelaar, Kjell Eikland, and Peter Notebart. lp_solve, May 2004.

[BLB$^+$13] R. Basmadjian, G. Lovasz, M. Beck, H. D. Meer, X. Hesselbach-Serra, J. F. Botero, S. Klingert, M. P. Ortega, J. C. Lopez, A. Stam, R. V. Krevelen, and M. D. Girolamo. A Generic Architecture for Demand Response: The ALL4green Approach. In *2013 International Conference on Cloud and Green Computing*, pages 464–471, September 2013.

[BMN$^+$07] Richard E. Brown, Eric R. Masanet, Bruce Nordman, William F. Tschudi, Arman Shehabi, John Stanley, Jonathan G. Koomey, Dale A. Sartor, and Peter T. Chan. Report to Congress on Server and Data Center Energy Efficiency: Public Law 109-431. Technical Report LBNL-363E, 929723, Ernest Orlando Lawrence Berkeley National Laboratory, August 2007.

[BRPC08] Christian Belady, Andy Rawson, John Pfleuger, and Tahir Cader. Green Grid Data Center Power Efficiency Metrics: PUE and DCIE. Technical report, Green Grid, 2008.

[Bun16] Bundesnetzagentur. Vorschlag aggregator-modell. online publication, 2016. Url: `https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Energie/Unternehmen_Institutionen/VortraegeVeranstaltungen/Aggregator_Modell_606.pdf;jsessionid=03755CA04B7603973C29122293D4C464?__blob=publicationFile&v=1`, Accessed: 2018-07-29.

[BW] Demand Side Management Pilotprojekt Baden-Wuerttemberg. Steckbrief:regelleistungsmarkt. website. Url: `http://www.dsm-bw.de/erloese-erzielen/markt-fuer-regelleistung/akteure/regelleistungsmarkt/`, Accessed: 24.07.2018.

[BZBK+16] Paolo Bertoldi, Paolo Zancanella, Benigna Boza-Kiss, European Commission, Joint Research Centre, and Institute for Energy and Transport. Demand response status in EU Member States. Technical report, Publications Office, Luxembourg, 2016. OCLC: 960394344.

[CAB+18] Tudor Cioara, Ionut Anghel, Massimo Bertoncini, Ioan Salomie, Diego Arnone, Marzia Mammina, Terpsichori-Helen Velivassaki, and Marcel Antal. Optimized flexibility management enacting Data Centres participation in Smart Demand Response programs. *Future Generation Computer Systems*, 78:330–342, January 2018.

[Cen] IBM Knowledge Center. Loadleveler job states. `https://www.ibm.com/support/knowledgecenter/en/SSFJTW_5.1.0/com.ibm.cluster.loadl.v5r1.load100.doc/am2ug_jobstate.htm`. Accesed: 2018-07-29.

[CNRB13] Rodrigo N. Calheiros, Marco A. S. Netto, César A. F. De Rose, and Rajkumar Buyya. EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of Cloud computing applications. *Software: Practice and Experience*, 43(5):595–612, 2013.

[CRB$^+$11] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, January 2011.

[CSP05] Kihwan Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(1):18–28, January 2005.

[dEA] Bundesverband der Energie-Abnehmer. Industriestrompreise (inklusive stromsteuer) in deutschland in den jahren 1998 bis 2018 (in euro-cent pro kilowattstunde). Url: `https://de.statista.com/statistik/daten/studie/252029/umfrage/industriestrompreise-inkl-stromsteuer-in-deutschland/`, Accessed: 26.07.2018.

[DoE06] U.S. Department of Energy. Benefits of Demand Response in Electricity Markets and Recommendations for Achieving them. Technical report, U.S. Department of Energy, February 2006.

[DWF16] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. Data Center Energy Consumption Modeling: A Survey. *IEEE Communications Surveys & Tutorials*, 18(1):732–794, 2016.

[ECLV12] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Understanding the future of energy-performance trade-off via DVFS in HPC environments. *Journal of Parallel and Distributed Computing*, 72(4):579–590, April 2012.

[EKR02] E. N. Mootaz Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy-Efficient Server Clusters. In *Power-Aware Computer*

Systems, volume 2325, pages 179–197, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[FWB07] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power Provisioning for a Warehouse-sized Computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 13–23, New York, NY, USA, 2007. ACM.

[GB11] S. K. Garg and R. Buyya. NetworkCloudSim: Modelling Parallel Applications in Cloud Simulations. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 105–113, December 2011.

[GGMP12] Girish Ghatikar, Venkata Ganti, Nance Matson, and Mary Ann Piette. Demand Response Opportunities and Enabling Technologies for Data Centers: Findings From Field Studies. Technical Report LBNL–5763E, 1174175, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), August 2012.

[GGWP14] Majid Ghasemi-Gol, Yanzhi Wang, and Massoud Pedram. An optimization framework for data centers to minimize electric bill under day-ahead dynamic energy prices while providing regulation services. In *Green Computing Conference (IGCC), 2014 International*, pages 1–9. IEEE, 2014.

[Gov07] U.S. Government. Energy Independence and Security Act, December 2007.

[GPF$^+$09] Girish Ghatikar, Mary Ann Piette, Sydny Fujita, Aimee McKane, Junqiao Han Dudley, Anthony Radspieler, K.C. Mares, and Dave Shroyer. Demand Response and Open Automated Demand Response Opportunities for Data Centers. Technical Report LBNL-3047E, 981725, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), December 2009.

[GTGB14] Saurabh Kumar Garg, Adel Nadjaran Toosi, Srinivasa K. Gopalaiyengar, and Rajkumar Buyya. SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter. *Journal of Network and Computer Applications*, 45:108–120, October 2014.

[HC14] Ralph Hintemann and Jens Clausen. Rechenzenten in Deutschland: Eine Studie zur Darstellung der wirtschaftlichen Bedeutung und der Wettbewerbssituation. *Studie im Auftrag des Bundesverbandes Informationswirtschaft, Telekommunikation und neue Medien eV (BITKOM), Berlin*, 2014.

[HF13] Ralph Hintemann and Klaus Fichter. Server und rechenzentren in deutschland im jahr 2012. online publication, 2013. Url: `https://www.borderstep.de/wp-content/uploads/2014/07/Kurzbericht_Rechenzentren_in_Deutschland_2012__09_04_2013.pdf`, Accessed: 2018-07-29.

[HFH$^+$09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[HK03] Chung-Hsing Hsu and Ulrich Kremer. The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, PLDI '03, pages 38–48, New York, NY, USA, 2003. ACM.

[(HL16] Höchstleistungsrechenzentrum Stuttgart (HLRS). Lesefassung der hlrs entgeltordnung 2017. online publication, 2016. Url: `https://www.hlrs.de/fileadmin/sys/public/solution_services/system_access/legal_requirements/HLRS_Entgeltordnung_Lesefassung_2017.pdf`, Accessed: 2018-07-29.

[HTHW16] Georg Hager, Jan Treibig, Johannes Habich, and Gerhard Wellein. Exploring performance and power properties of modern multi-core chips via simple machine models. *Concurrency and Computation: Practice and Experience*, 28(2):189–210, 2016.

[ind] Global warming: Data centres to consume three times as much energy in next decade, experts warn. `http://www.independent.co.uk/environment/global-warming-data-`

centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html. Accessed: 2018-01-28.

[Int] Intel. Techical resources: Intel xeon processors. https://www.intel.com/content/www/us/en/processors/xeon/xeon-technical-resources.html. Accessed: 2018-07-29.

[Jai10] Anil K. Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666, June 2010.

[KBK12] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, December 2012.

[LWC$^+$13] Zhenhua Liu, Adam Wierman, Yuan Chen, Benjamin Razon, and Niangjun Chen. Data center demand response: Avoiding the coincident peak via workload shifting and local generation. *Performance Evaluation*, 70(10):770–791, October 2013.

[MWW12] D. Meisner, J. Wu, and T. F. Wenisch. BigHouse: A simulation infrastructure for data center systems. In *2012 IEEE International Symposium on Performance Analysis of Systems Software*, pages 35–45, April 2012.

[Net16] Cisco Visual Networking. Cisco global cloud index: Forecast and methodology, 2015-2020. white paper. online publication, 2016. Url: https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf, Accessed: 01.02.2018.

[NEX] NEXTKraftwerke. Was ist sekundaerregelleistung (srl)? website. Url: https://www.next-kraftwerke.de/wissen/regelenergie/sekundaerreserve, Accessed: 24.07.2018.

[ns2] The network simulator -ns-2. https://www.isi.edu/nsnam/ns/. Accessed: 2018-07-29.

[OPPF10] Simon Ostermann, Kassian Plankensteiner, Radu Prodan, and Thomas Fahringer. GroudSim: An Event-Based Simulation Framework for Computational Grids and Clouds. In *Euro-Par 2010 Par-*

allel Processing Workshops, Lecture Notes in Computer Science, pages 305–313. Springer, Berlin, Heidelberg, August 2010.

[QKP^{+}09] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma. Towards autonomic workload provisioning for enterprise Grids and clouds. In *2009 10th IEEE/ACM International Conference on Grid Computing*, pages 50–57, October 2009.

[RLSdS11] Barry Rountree, David K. Lowenthal, Martin Schulz, and Bronis R. de Supinski. Practical performance prediction under Dynamic Voltage Frequency Scaling. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8. IEEE, IEEE, July 2011.

[RRT^{+}08] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. No "Power" Struggles: Coordinated Multi-level Power Management for the Data Center. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, pages 48–59, New York, NY, USA, 2008. ACM.

[Sia14] Pierluigi Siano. Demand response and smart grids—A survey. *Renewable and Sustainable Energy Reviews*, 30:461–478, February 2014.

[SKGH12] Thomas Schulze, Marcus Kessel, Giovanni Giuliani, and Xavier Hesselbach. D6.2 all4green simulation tool description. online publication, 2012. Url: `http://all4green-project.gfi.es/deliverables`, Accessed: 2018-07-29.

[SS11] STULZ IT Cooling Solutions and Services. Lösungen für rz-klimatisierung. online publication, 2011. Url: `https://www.hamburg.de/contentblob/2865410/0a544e4ea228403ac46e493b6f75c148/data/vortrag-pfleiderer-14-04-2011.pdf;jsessionid=995FAEF0421A5C2190E4D5C941076856.liveWorker2`, Accessed: 2018-07-29.

[SWA^{+}15] Hayk Shoukourian, Torsten Wilde, Axel Auweter, Arndt Bode, and Daniele Tafani. Predicting Energy Consumption Relevant Indicators of Strong Scaling HPC Applications for Different Compute Resource Configurations. In *Proceedings of the Symposium on High Performance Computing*, HPC '15, pages 115–126, San Diego, CA, USA, April 2015. Society for Computer Simulation International.

[SWAB14] Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. Monitoring Power Data: A first step towards a unified energy efficiency evaluation toolset for HPC data centers. *Environmental Modelling & Software*, 56:13–26, June 2014.

[SWHB17] Hayk Shoukourian, Torsten Wilde, Herbert Huber, and Arndt Bode. Analysis of the efficiency characteristics of the first High-Temperature Direct Liquid Cooled Petascale supercomputer and its cooling infrastructure. *Journal of Parallel and Distributed Computing*, 107:87–100, September 2017.

[SWLB17] H. Shoukourian, T. Wilde, D. Labrenz, and A. Bode. Using Machine Learning for Data Center Cooling Infrastructure Efficiency Prediction. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 954–963, Lake Buena Vista, FL, USA, May 2017.

[tC12] European Parliament and the Council. Directive 2012/27/EU of the European Parliament and the Council, November 2012.

[TDC$^+$14] C.-J. Tang, M.-R. Dai, C.-C. Chuang, Y.-S. Chiu, and W.S. Lin. A load control method for small data centers participating in demand response programs. *Future Generation Computer Systems*, 32:232–245, March 2014.

[TOPa] TOP500. Highlights: June 2012. `https://www.top500.org/lists/top500/2012/06/highlights/`. Accesed: 2018-07-29.

[TOPb] TOP500GREEN. Green500 list - june 2017. `https://www.top500.org/green500/list/2017/06/?page=3`. Accesed: 2018-07-29.

[TT13] Bogdan Marius Tudor and Yong Meng Teo. On Understanding the Energy Consumption of ARM-based Multicore Servers. In *Proceedings of the ACM SIGMETRICS/International Conference on*

Measurement and Modeling of Computer Systems, SIGMETRICS '13, pages 267–278, New York, NY, USA, 2013. ACM.

[WLLMR14] A. Wierman, Z. Liu, I. Liu, and H. Mohsenian-Rad. Opportunities and challenges for data center demand response. In *International Green Computing Conference*, pages 1–10, November 2014.

[YLZ14] Jianguo Yao, Xue Liu, and Chen Zhang. Predictive Electricity Cost Minimization Through Energy Buffering in Data Centers. *IEEE Transactions on Smart Grid*, 5(1):230–238, January 2014.

Appendix

A. Additional Information to the Original Implementation of DCSim

A.1. Service Implementation

The `Service` class is located in the *eu.a4g.dcSim.dcConfiguration* package. It implements the service component within the DCSim framework. To do so, the class stores several attributes which are supposed to capture the configuration of a service (SLA information, start time, host DC), provide information for explicit identification of a service (id, name), and different information that are required due to the current implementation (information whether a service was modified, scheduling interval of the scheduler, list of VMs, status). As a service is also an energy consumer, the class also provides an `update`-method. The purpose of it is to check whether the SLA (see Subsection 5.3.4 for more information) parameters have changed, to check whether the service has to be restarted during the next scheduling interval, and to maintain the statuses of the VMs of a service.

When a service restarts during the next scheduling interval, the method requests an allocation of resources to the service. To do this the private `getITService-AllocationAdvice`-method, which momentarily returns -1 as a default value, is called. For the case that a service is in the status *off*, the method ensures that all VMs also are switched to the *off* status. In contrast, when the service is currently in the status *running* or *paused*, the current utilization and the status of the last VM in the list of VMs is checked. Which adjustments the `update`-method then implements depends on the values of these two variables. When the utilization value is zero and the VM is in the status *running*, all VMs are put into the *off* status. For the case that the utilization value is not zero and the VM is in the status *off*, the method switches a certain amount of VMs into the *running* status. The number of switched VMs depends on the current SLA availability parameter. An availability parameter of zero causes a switch of

$(TotalNumberOfVMs/2) - 1$ VMs, whereas an availability parameter value of one causes a switch of $((TotalNumberOfVMs)/2) * 1.1) - 1$ VMs. Whenever some adjustments to the service, which affect the current simulation situation, are implemented, the `modified`-attribute of the instance is set to `true`.

Another important method that is provided by this class is the `changeSLA`-method. Via this method, a user can change all SLA parameters which are defined by the utilized SLA model (see Subsection 5.3.4). When the availability parameter is changed, the method adjusts the amount of running VMs accordingly. For changes of all other parameters, the method simply adjusts the corresponding attributes within the `VM` instances that are allocated to the service.

A.2. VM Implementation

The `VM` class, which is located in the *eu.a4g.dcSim.dcConfiguration* package, implements the concept of a VM. Therefore, it contains attributes that describe the configuration of a VM (million instructions per second, number of CPU cores, size of available RAM, available network bandwith, and available harddisk space). In addition, the class stores attributes that provide unique identification of a VM instance, capture the current status of a VM instance, and store time series data for the utilization of the VM. As possible adjustments to the parameters of a VM instance and the energy consumption of a VM is already captured by the `Server` and `Service` classes, there is no need to implement an `update`-method within the `VM` class as well.

A.3. Optimizer Implementation

The optimizer component is responsible for the optimal resource allocation within the DC. Originally, DCSim provides a simple implementation for this component optimizes the resource allocation to the services with a special focus on SLA compliance. Thus, it must be guaranteed at any time that the optimization process does not lead to any SLA violations. The simple optimizing algorithm works as follows [SKGH12]:

1. In the first step of the algorithm, the most scarce resource (where the resources are similar to the ones captured by the SLA performance metrics) in the system is identified.
2. In this step, all servers and services get sorted in descending order by the most scarce resource.
3. In this step the optimizer tries to allocate the services to the servers in the order of the sorted lists. Thereby, the optimizer starts with the first service in the list and loops through the list of servers until it has found a server that fulfills the resource requirements of the service. When it finds such a server the service is allocated to this server and the process is repeated with the next service in the list. In the case the there is no sufficient server, the optimization process fails.

This algorithm does not provide an optimal solution and is more likely to fail the more heterogeneous the hardware resources are in the considered system [SKGH12].

This optimization algorithm is implemented by the `Optimizer` class which is located in the *eu.a4g.dcSim* package. An instance of `Optimizer` provides two methods which are the `optimize`- and the `isFull`-method. The two methods basically implement the algorithm which is described above and return a `boolean`-value that indicates whether the algorithm was completed successfully or not. The only difference between the methods is that the `optimize`-method actually allocates resources to the services, whereas the `isFull`-method only checks whether the algorithm can be executed successfully but allocates no resources to the services.

A.4. ESF Implementation

The `ESF` class has several attributes that are necessary to implement the ESF model that is utilized in DCSim. These are the efficiency, the current charging level, the charging input, and the capacity. Another important attribute of this class is the status which can be any of the three values *charging*, *off*, or *discharging*. Furthermore, the class stores an `id` attribute for identification, the current power consumption, and historical time series data for the power consumption and utilization. Similar to the other energy consumer classes, the `ESF` class implements an `update`-method which contains the central logic of the class. In the case of this class, the `update`-method adjusts the charging level and current power consumption of the ESF. The method basically distinguishes between three cases, where each case corresponds to one status in which the ESF can be. Depending on the current status of the ESF, the update method performs different adjustments. When the ESF is currently in the *charging* or *discharging* status, the utilization and current power consumption is adjusted according to the model which is described in Subsection A.4.1. For the third case, when the ESF is currently in the status *off*, the current power consumption and utilization are set to 0.

A.4.1. ESF Power Model

When the ESF is currently in the *charging* status, the current power consumption of the ESF is considered to be equal to the specified charging input and the charging level of the ESF is calculated according to the following equation:

$$cL_{new} = \frac{((\frac{c*cL_{old}}{100}) + cI) * \frac{eff}{100} * \frac{sI}{3600}}{c*100} \tag{A.1}$$

Where cL_{new} is the new charging level, cL_{old} is the old charging level, c is the capacity of the ESF, cI is the charging input of the ESF, eff is the efficiency of the ESF, and sI is the scheduling interval. The utilization is set to 0 for this case. For the case that the ESF is currently in the *discharging* status, the current power consumption is set to the negative current overall power consumption of the DC and the charging level is calculated as follows:

$$cL_{new} = \frac{100 * (\frac{c*cL_{old}}{100} - p_{DC} * \frac{sI}{3600}}{c} \tag{A.2}$$

Where p_{DC} is the current overall power consumption of the DC. The utilization of the ESF is set to the combined utilization of HVAC and all servers.

A.5. Information Retriever

Basically, this class stores the monitoring data which are interesting for an analysis of the simulation and provides a well defined interface to the functionalities of a simulated DC. Each instance of this class is assigned to one distinct instance of the `DC` class. Therefore, the `InformationRetriver` class has one attribute that stores the assigned `DC` instance, one attribute that specifies the start time of the current simulation, and eight attributes that store time series data for the monitoring data. In the original version, the monitoring data included the server power consumption, the server utilization, the ESF power consumption, the ESF utilization, the ESF charging level, the HVAC power consumption, the HVAC utilization, and the inhouse temperature. All attributes of the class are defined as private instance attributes. The constructor takes only the assigned instance of `DC` as a parameter and initializes the `dc` attribute. In addition, the class provides several methods that can be used to retrieve simulation values and to change several parameters of a running simulation.

The `getCurrentDate`-method can be used to retrieve the current time of the simulation in the form of an instance of the class `SimpleDateFormat` To get time series data for all servers, the `getServer`-method can be used. It returns a list of three `Object` arrays, from which the first array contains the ids, the seconds array the current utilizations, and the third array the current power consumptions. In the same way, the `getVMs`-method return information about all VMs. In this case, the arrays contain information about the id, utilization, MIPS, RAM size, HDD size, bandwidth, and assigned server. Another method that follows the same structure is the `getServices`-method. It provides access to the id, assigned servers, and assigned VMs informations of a service. Alternatively, only a single service, which is provided as instance of `Service`, can be retrieved. Two additional methods, which are related to the services, are `getServiceDetails` and `getSLAforService`. The `getServiceDetails`-method takes a `String` with the service's id as parameter and returns a list of two `Strings`, where the first one contains the service's id and the second one contains either the name of the service or a message which indicates that the service was not found. As the name suggests, the `getSLAforService`-method provides access to the SLA specifications of a distinct service. This method also takes a service's id as input and provides an instance of the class `Map<String, String>`, which contains a mapping between all SLA parameter names and there corresponding values for the requested service, as a result. The `InformationRetriever` class does not only provide the possibility to retrieve a service's SLA information, but also to change them. For this purpose, the `changeSLA`-method can be used. This method changes the SLA parameters of a service according to the input parameters which are similar mapping as the `getSLAforService` method provides as a result and a service's id as input parameters. The fifth method, which is called `getService-State`, that is related to services returns the current status of a service.

The `InformationRetriever` class cannot only be used to access and adjust parameters of a DC's components, but also the system wide parameters of a DC. To change the temperature and the use of the ESFs, one can use the `change-DCParam`-method. This method takes a list of one or more `String` 2-tuples (e.g., *("AC", "20.0")* to change the temperature or *("ESF", "charging")* to change the status of the ESFs) which define the requested adjustments. Furthermore, getter-methods for the start time, PUE, total costs, and ESF status is provided. The start time of the simulation can also be changed via the `setStartTime`-method.

For analysis purposes, the class also provides access to the time series data of the monitored values. This is done via a get-method for each time series. Before the corresponding methods return the time series data, they call the `updateData`-method which updates the stored time series according to the current simulation status.

A.6. Additional Information on the Server Implementation

To determine the current utilization of the server, the method checks for each `VM` instance in the list of hosted VMs whether the VM is in the status *running*. When this is the case, it is checked whether the utilization of the VM is positive at the current timestamp of the simulation. For the case that this condition does not hold, the current server utilization is not changed. In contrast, when the condition does hold, the utilization fraction, which is caused by this VM, is calculated and added to the current server utilization. The utilization fraction of the VM is calculated according to the following equation:

$$uFraction_{vm_i} = \frac{mips_{vm_i} * u_{vm_i}}{mips_{server}} * \frac{pes_{vm_i}}{pes_{server}} \tag{A.3}$$

Where $uFraction_{vm_i}$ is the utilization fraction of VM i, u_{vm_i} is the current utilization of VM i, $mips_{server}$ is the amount of million instructions the server can execute per second, $mips_{vm_i}$ is the amount of million instructions VM i can handle per second, pes_{server} specifies the amount of CPU cores of the server, and pes_{vm_i} expresses the amount of CPU cores of VM i. Consequently, the current utilization of a server can be calculated as follows:

$$u_{server} = \sum_{i=1}^{N_{vm}} uFraction_{vm_i} \tag{A.4}$$

Where u_{server} is the current utilization of a server, and N_{vm} is the amount of VMs that the server currently hosts.

A.7. Additional Information on the HVAC Implementation

Within the implementation of the HVAC component, adjustments of the current building temperature are determined through a simple if-else tree:

$$t_{building} = \begin{cases} t_{building} + \Delta_{interval}t & \text{if } t_{building} < t_{setpoint} \\ t_{building} - \Delta_{interval}t & \text{if } t_{building} > t_{setpoint} \end{cases} \tag{A.5}$$

Where $t_{building}$ is the current temperature within the DC building, $\Delta_{interval}t$ is the temperature change per interval, and $t_{setpoint}$ is the currently specified temperature setpoint of the HVAC system. Thereby, $\Delta_{interval}t$ is a prespecified constant value. The current utilization of the HVAC system is calculated on the basis of the current overall utilization of the servers:

$$u_{HVAC} = u_{servers} * (1 - \frac{t_{building} + t_{min}}{100}) \tag{A.6}$$

Where u_{HVAC} is the current utilization of the HVAC system, $u_{servers}$ is the current overall utilization of all servers, and t_{min} is the minimal reachable temperature.

A.7.1. HVAC Energy Consumption Model

The implementation of the STULZ model is based on the `IStulzPowerModel` interface, the abstract `PowerModelSpecHVAC` class, and the `PowerModelSpec-Stulz` class which are all located in the *eu.a4g.dcSim.powerModels* package. The `IStulzPowerModel` defines a getter- and a setter-method for all additional components that are considered. Within the classes that implement this interface, these methods are intended to return the fraction of the energy consumption that is caused by the corresponding component. As the name already suggests, the `PowerModelSpecHVAC` is inherited from the `PowerModelSpec` class which was descried earlier. Therefore, it also defines the abstract `getPowerData`-method. In addition, it provides a private instance attribute `MAX_WATTAGE` and a getter- and setter-method for this attribute. The actual implementation of the model is located within the `PowerModelSpecStulz` class which is inherited from the `PowerModelSpecHVAC` class and implements the `IStulzPowerModel` interface. As it thus, has to implement all methods that are defined by the interface, it stores two attributes for each of the additional components. One attribute for the actual fraction of the energy consumption of each component and one attribute for a change factor of that component. An exception to this is the IT power, where the class stores the fraction, a maximum IT power value, and a minimum IT power value. The change factors for each of the components describes the influence that the IT (server) utilization rate has on the energy consumption of the corresponding components. A value of one means that the energy consumption changes proportional to the IT utilization value, whereas a value of zero means that it does not change at all. The class provides two constructors, where one constructor takes all energy fraction values as input parameters and initializes the change factors with default values and the other constructor takes both, the energy fraction values and the change factors as input values. To obtain the current modeled energy consumption, the class provides the `getAdditionalEnergyConsumption`- and the `getPowerData`-method. Where the first one is used for the energy consumption in the average use case and the second one can be used for use cases where the temperature inside the DC moves outside the average corridor.

The `getAdditionalEnergyConsumption`-method is implemented in a way so that the additional energy consumption, which includes the energy consumption of lighting, generators, non-interruptible power supplies, chillers, humidifiers, and the air conditioning, of a DC can be calculated based on the current IT utilization and a linear- or a cubic model. The linear and cubic models are implemented in the classes `PowerModelLinear` and `PowerModelCubic`. The total additional energy consumption is then calculated as the sum of the additional energy consumptions of each of the components which is calculated based on the specified model.

B. Additional Information to the Adjusted Implementation of DCSim

B.1. Structure of the Configuration File

The configuration file that is used for the simulation has to match the following doctype definition:

```
<!DOCTYPE SimulationData
[
<!ELEMENT SimulationData(SimulationSetup,SimLength,SimStartTime,WorkloadTrace-
File,DREventTraceFile,PUETraceFile,EvaluationDatabase,WekaPowerModelFile,
DataCentre)>
<!ELEMENT SimulationSetup(SchedulingInterval,SecondsPerSimulationTimestep,
SLACostExponent,SuperMUCMode,SolverTimeout)>
<!ELEMENT SchedulingInterval(#PCDATA)>
<!ELEMENT SecondsPerSimulationTimestap(#PCDATA)>
<!ELEMENT SuperMUCMode(#PCDATA)>
<!ELEMENT SolverTimeout(#PCDATA)>
<!ELEMENT SimLength(#PCDATA)>
<!ELEMENT SimStartTime(#PCDATA)>
<!ELEMENT WorkloadTraceFile(#PCDATA)>
<!ELEMENT DREventTraceFile(#PCDATA)>
<!ELEMENT EnergyPriceTraceFile(#PCDATA)>
<!ELEMENT PUETraceFile(#PCDATA)>
<!ELEMENT EvaluationDatabase(#PCDATA)>
<!ELEMENT WekaPowerModelFile(#PCDATA)>
<!ELEMENT DataCentre(Name,PUE,ServerConfiguration,HVAC)>
<!ELEMENT Name(#PCDATA)>
<!ELEMENT PUE(#PCDATA)>
<!ELEMENT ServerConfiguration(NbOfServer,MIPS,PES,RAM,BW,Storage)>
<!ELEMENT NbOfServer(#PCDATA)>
<!ELEMENT MIPS(#PCDATA)>
<!ELEMENT PES(#PCDATA)>
<!ELEMENT RAM(#PCDATA)>
<!ELEMENT BW(#PCDATA)>
<!ELEMENT Storage(#PCDATA)>
<!ELEMENT HVAC(StandardTemp)>
<!ELEMENT StandardTemp(#PCDATA)>
]>
```

Where the parameters are supposed to specify the following information:

SchedulingInterval: This parameter specifies the length (in simulation time) of the time period for which the workload is scheduled during one call of the scheduler.

SecondsPerSimulationTimestep: This parameter defines the amount of real time seconds that are incorporated in one step of simulation time. Thus, when this value is set to 1, each step in simulation time corresponds to one second in real time.

SuperMUCMode: To be parsed correctly, this parameter should either contain "on" or "off". Corresponding to this, the simulator will run in *superMUC-Mode* or not.

SolverTimeout: This parameter specifies the timeout (in real time seconds) of the linear optimization solver.

SimLength: This parameter specifies the intended length of the simulation run in simulation time.

SimStartTime: This parameter specifies the start date of the simulation run as a date string. An example date string could look like this: "2014-01-01T00:00:00".

WorkloadTraceFile: This parameter specifies the path to the *.csv* file that contains the workload trace, which should be used for the simulation run.

DREventTraceFile: This parameter specifies the path to the *.csv* file that contains the DR event trace, which should be used for the simulation run.

EnergyPriceTraceFile: This parameter specifies the path to the *.csv* file that contains the energy price trace, which should be used for the simulation run.

PUETraceFile: This parameter specifies the path to the *.csv* file that contains the PUE trace, which should be used for the simulation run.

EvaluationDatabase: This parameter specifies the path to the location where the database file, which contains the monitored data, should be placed.

WekaPowerModelFile: This parameter specifies the path to the file that contains the WEKA linear regression models, which can be used by a weka based server power consumption model.

PUE (in DataCentre element): This static PUE value will be used when no PUE trace is specified.

MIPS: This parameter specifies the amount of millions of instructions one server of the DC can process.

PES: This parameter specifies the amount of CPUs that a sever of the DC has.

RAM: This parameter specifies the size of the random access memory (in MB) of each server of the DC.

BW: This parameter specifies the network bandwith (in bit) of each server of the DC.

Storage: This parameter specifies the size of the harddisk space that each server of the DC can use.

The parameters `MIPS`, `PES`, `RAM`, `BW`, `Storage`, and `StandardTemp` were necessary for the original version of DCSim and are still read from the configuration for the case that future developers would like to use them in the implementation. However, in the current version of the simulation, these values are not actively used.

B.2. Event Handling

B.2.1. Event Queue and Events

The event queue concept is implemented by the `EventQueue` class. The events that are scheduled within the event queue are represented by the `Event` class. Every instance of the `Event` class contains a variable that specifies the point in simulation time at which the event is scheduled and a variable that indicates the type of the event. The different types of events are captured by the `EventType` enum. In the adjusted version of DCSim there are nine different event types, which are namely `JOB_START`, `JOB_FINISH`, `JOB_PAUSE`, `JOB_RESTART`, `JOB_SUBMISSION`, `INCREASE_COOLING_SETPOINT`, `DECREASE_COOLING_SETPOINT`, `SERVER_UPDATE`, and `DR_REQUEST`. As each type of event might affect a different type of system component (for example a `JOB_START` event affects an instance of the `BatchJob` class), in the current implementation there exist the `JobEvent`-, `ServerEvent`-, `DRRequestEvent`, and `CoolingEvent` classes which are all inherited from the `Event` class. All these classes contain an additional variable that specifies the component that is affected by the corresponding event. This architecture is designed to make it easy for future developers to design own event classes and to introduce them into the existing system. To add a new event type, future developers can simply add the type to the `EventType` enum and define a new class, which is inherited from the `Event` class, that represents the new event. The `EventQueue` class implements the functionality that is necessary to schedule, reschedule, unschedule, and retrieve scheduled events. To do so, all events of one type are maintained in a separate instances of `HashMap`. Each map takes an simulation timestep value as the key and has an `ArrayList`, which stores instances of the `Event` class, as values. When a new instance of `EventQueue` is created via the constructor, the constructor automatically creates such a map for each event type that is listed in the `EventType` enum. This mechanism ensures that the `EventQueue` class does not have to be touched by a developer that likes to add a new event type as the new event type is automatically included. The class also provides a constructor that takes a list of batch jobs as a parameter and already schedules a submission event for each job in this list during the initialization of the event queue.

To schedule an event, a developer can either use the `schedule`- or the `reschedule`-method. The difference between these two methods is that the `schedule`-method simply adds the event, which is passed as a parameter, to the list of the corresponding map that is mapped to the simulation timestep, which is specified by the event. In contrast, the `reschedule`-method checks whether the event, which is passed as a parameter, already existed in the event queue and if yes removes it before it schedules the event at the new timestamp, which is also passed as a parameter. Thereby, the `reschedule`-method uses the natively implemented `equals`-method of the `Object` class. Thus, the event is only removed if it is really the same instance with the same memory address. To ensure that this mechanism works properly is also the reason why the instances of the `BatchJob` class store three instances of `Event`. As the `reschedule`-method works exactly as the `schedule`-method for the case that the passed event is not found in the event queue, it is recommended to future developers that they favor the use of the `reschedule`-method. The event queue also provides the possibility to unschedule events via the use of the `unschedule`-method. Similar as for the `reschedule`-method, the `equals`-method of the `Object` class is used to detect the event that should be removed within the event queue. To ensure that the maps grow not too big for long simulations, the `EventQueue` class provides a `removeKey`-method which removes the simulation timestep, which is passed as a parameter, from all maps in the event queue. To do this after every step of the simulation can improve the performance of the simulation framework significantly. Within the simulation framework, the event queue is stored and maintained within the `DC` class, which also provides methods to schedule and retrieve events from the event queue.

B.2.2. Event Handling and Resource Allocation

The actual event handling and resource allocation process is implemented within the `EventHandler` class. The `update`-method of an instance of this class is called at the beginning of each call of the `updateJobAllocation`-method of the `DC` class and is responsible for the proper handling of all scheduled events and to allocate servers to each job that is supposed to be executed. For its operation, it maintains two lists of servers, where one lists stores all currently occupied servers and the other list stores currently idle servers. Furthermore, it maintains separate lists for all currently running jobs, all currently paused jobs, and all already finished jobs. This split into several lists improves the performance of the simulation framework, as this enables the event handler to only loop through the jobs/servers that might be relevant for a certain operation. In addition, every instance of the `Event-Handler` class maintains a mapping between server ids and job ids, where the server ids are used as the key to store the id of the job that is currently executed by the corresponding server. This mapping is used to speed up the deep copy process of a `DC` instance. In order to access the events in the event queue, the class also has a variable that stores the `DC` instance to which the event handler belongs. The `EventHandler` class implements a separate processing method for of each of the currently used event types. Each of these methods is called, when the `update`-method of the `EventHandler` class is called and there is at least one event of the corresponding type scheduled at the current timestep. In addition, the `update`-method calls the `updateRunningJobs`-method, which ensures that for all currently running and paused jobs the `elapseOneSimulationStep`-method is called. Within the corresponding methods the different events are handled as follows:

`JOB_PAUSE` **Events:** For each pause event, when the related job is not already ressscheduled (indicated by the `alreadyRescheduled` variable, the status of the related job is set to `PAUSED` and the job is moved from the running job list to the paused job list. Otherwise, the status is set to `RESCHEDULED` and the job is moved from the running job list to the scheduled job list. Afterwards, the status of all servers that were assigned to the job is changed to `IDLE`, the server is moved from the occupied server list to the idle server list, and a `SERVER_UPDATE` event is scheduled for each server that was assigned to the job.

`JOB_FINISH` **Events:** For each finish event, the status of the affected job is adjusted to `FINISHED`, the actual finish time of the job is set to the current time (this also triggers the calculation of the delay of the job), and the job is moved to the list of the `EventHandler` instance which contains all finished jobs. Furthermore, the status of all servers that were assigned to the job is set to `IDLE`, the server is moved back to the idle server list, and a `SERVER_UPDATE` event is scheduled for each of the assigned servers.

`JOB_SUBMISSION` **Events:** For each job submission event, the batch job that belongs to the event is submitted to the scheduler of the DC. When the start time that is currently scheduled for the job is after the next scheduler call, this is done via the call of the `submitParsedJob`-method of the `DC` class. Otherwise, it is done by the call of the `submitAffectedSubmittedJob`, as the job cannot be started at its originally scheduled start time.

`JOB_START` **Events:** For each start event it is first checked whether the linked batch job is in the status `SCHEDULED`. When this is not the case, nothing is done for this event. In contrast, when it is the case, it is checked whether there are currently enough idle servers that can be assigned to the job. When this is not the case, an error message is printed to the console, the job's status is set back to `SUBMITTED`, and the job is moved to the `affectedSubmittedJobs` list. In addition, the finish even and the start event are unscheduled for the job. In the case that there are enough idle servers available, the requested amount of servers is assigned to the job, the statuses of the job and each of the assigned servers is adjusted, and a `SERVER_UPDATE` event is scheduled for each of the assigned servers. In addition, the job is added to the list of the `EventHandler` instance that contains the currently running jobs and is removed from the list in the `Scheduler` instance that contains all currently scheduled jobs. Furthermore, the finish event of the job is rescheduled.

`JOB_RESTART` **Events:** For each restart event basically, besides one minor difference, the same actions are taken as for the start events. One difference is that it is checked whether the affected jobs are in the `PAUSED` status instead of the `SCHEDULED` status.