POLITECNICO DI MILANO

Master of Science in Automation And Control Engineering
Department of Electronics, Information and Bio-Engineering

Object-oriented modelling of data centre servers for local controls and system-level studies

Supervisor
Prof. Alberto LEVA

Presented by
Hafsa FAROOQI, 796568

Academic Year 2014-2015

. . . this page intentionally left blank . . .

Contents

List of Figures

List of Code Snippets

Summary

A major issue with data centres is the tremendous amount of energy consumed ranging from a few kW for a rack of servers in a closet to several tens of MW for large facilities.

Some facilities have power densities more than 100 times that of a typical office building .It can be estimated from the fact, that at present, in United States, it takes 34 power plants, each capable of producing 500 MW of electricity, to power all the data centres in operation and by 2020, the nation will need another 17 similarly sized power plants to meet projected data centre energy demands as economic activity becomes increasingly digital. More generally in a large data centre, the amount of electricity needed to run it is similar to that of a small town.

The major aspect to be understood in this context is the main consumers of energy in a Data Centre.There are two main sinkers of energy, firstly the power needed to run the servers and secondly the power needed to cool them down as the temperature increase caused by the high energy consumption can prove fatal to the life of servers and lead to inefficiencies in carrying out the commands that are to be executed by the servers. Another added burden is due to the need of redundancy of all these equipments to save the data centre from complete shutdown in case of some interruption, as the complete shutdown just for a short while can lead to the loss of billions of dollars.

In recent years, a lot has been done by the computer science field using methodologies such as Dynamic Voltage Frequency Scaling (DVFS) and other techniques but not as much has been done from the point of view of system modelling and application of classical control techniques for the optimization of energy and temperature control at local as well as central level. The main objective of the thesis was to develop system models at the server levels taking into account all the important aspects such as number of instructions to be executed, DVFS and the energy consumed. It was followed by its temperature control by implementation of the discrete event controller which was successfully achieved. Furthermore, the the complex modelling of the racks and hot and cold aisles using the basic principles of thermodynamics was carried and then integrated to form a complete model of the data centre.

This thesis is part of a long-time research, aimed at an integrated control of the data centre machines, and the air conditioning system. A previous thesis [1]

addressed the modelling of airflow and conditioning, having however the servers just described as exogenous sources of thermal power. This permitted to set up some system-level control, including a very simple load re-allocation mechanism, but not to appreciate realistically enough the consequences of said control on the behaviour of the single server as for the accomplishment of its computational demand. This work, on the contrary, focuses on describing the server with the simplest model capable of fulfilling the need just mentioned, and previously left open.

The usefulness of the proposed model for the intended purpose is demonstrated by employing that model to set up local controls for the servers' thermal behaviour. The integration with previous results, so as to obtain a complete data centre model, is left to future works. Nonetheless, the simulation performance of the proposed models is high enough to allow to state that its use for "large-scale" system-level studies will be successful.

Sommario

Un elemento importante della gestione dei data centre è il loro gande consumo energetico, che puó variare da pochi kW per un singolo rack di server fino a diverse decine di MW per grandi installazioni.

Alcuni data centre hanno densità di potenza superiori di 100 volte quella di un tipico edificio per uffici. Per esempio, dal momento che attualmente negli Stati Uniti ci vogliono 34 centrali elettriche, ognuna in grado di produrre 500 MW di potenza, per alientare tutti i data center in funzione, e dato che entro il 2020 la nazione avrà bisogno di altri 17 impianti di dimensioni simili per soddisfare la domanda di energia prevista per i data centre di futura installazione, poiché le attività economiche diverranno sempre più digitali. Più in generale, in un grande data centre, la quantità di energia elettrica necessaria al suo funzionamento è simile a quella di una piccola città.

L'aspetto principale da considerare in questo contesto consiste nel comrpendere e analizzare quali sono i principali consumatori di energia in un data centre. A tal proposito due sono i principali consumi di energia, in primo luogo la potenza necessaria per far funzionare i server e in secondo luogo la potenza necessaria per raffreddarli, dal momento che l'aumento di temperatura causato dall'elevato consumo energetico può rivelarsi fatale per la vita del server e portare a inefficienze nello svolgimento dei compiti che esso deve eseguire. Un'altra difficoltà viene dalla necessità di ridondanza di tutti queste apparecchiature, per salvaguardarere il data centre da un completo shutdown in caso di interruzione di alcuni servizi, visto che l'arresto completo solo per un breve periodo può portare alla perdita di miliardi di dollari.

Negli ultimi anni, molto è stato fatto dal settore informatico utilizzando metodologie come Dynamic Voltage Frequency Scaling (DVFS) e altre tecniche, ma non altrettanto è stato fatto dal punto di vista della modellazione del sistema e dell'applicazione di tecniche di controllo classiche per la ottimizzazione del controllo dell'energia e della temperatura sia a livello locale che a livello centrale. Il principale obiettivo della tesi è quello di sviluppare modelli di sistema a livello di server, tenendo conto di tutti gli aspetti importanti quali il numero di istruzioni da eseguire, la gestione del DVFS e l'energia consumata. Tale obiettivo è seguito dall'implementazione di un controllo di temperatura centrato su un regolatore event-based, che è stato realizzato con successo.

Questa tesi è parte di una ricerca di lungo periodo, volta a un controllo integrato delle macchine del data center e dell'impianto di climatizzazione. Una tesi precedente [1] ha affrontato la modellazione dei flussi d'aria e del condizion-

amento, descrivendo tuttavia i server come fonti esogene di potenza termica. Ciò ha consentito di delineare delle strategie di controllo a livello di sistema, comprendendo un meccanismo molto semplice di riallocazione del carico, ma non di apprezzare abbastanza realisticamente le conseguenze di tale controllo sul comportamento del server singolo quanto al soddisfacimento della sua domanda di carico computazionale. Questo lavoro, al contrario, si concentra sulla descrizione del server con il modello più semplice in grado di soddisfare la necessità appena citata, e precedentemente lasciata aperta.

L'utilità del modello proposto per lo scopo sopra descritto è dimostrata utilizzando quel modello per mettere a punto dei controlli locali per il comportamento termico dei server. L'integrazione con risultati precedenti, così da ottenere un modello completo del data centre, è lasciato a lavori futuri. Tuttavia, le prestazioni di simulazione dei modelli proposti sono sufficientemente elevate da permettere di affermare che il loro uso per studi 'large-scale" a livello di sistema avrà successo.

Chapter 1

Introduction

1.1 Introduction

In today's technical world, the term DATA CENTRES or in general SERVER ROOMS need no introduction. They are the most essential parts of all the big and small companies operating today. In order to store information and run on and offline servers and networks,all the companies need big or small data centres depending on the intensity of the tasks to be accomplished by each company . The importance of these centres can be ascertained from the fact, that companies like Google, have entire buildings under lock and key dedicated to their advanced data storage techniques. Figure 1.1 shows a typical data centre architecture.

Historically speaking, the basic existence of these data centres can be attributed to the need of fast Internet connectivity and nonstop operation by the companies to deploy systems and establish a presence on the internet. Many companies started building very large facilities, called Internet Data centres(IDC), which provided businesses with a range of solutions for system deployment and operation.

The main issue with the data centres is the tremendous amount of energy consumed ranging from a few kW for a rack of servers in a closet to several tens of MW for large facilities. Some facilities have power densities more than 100 times that of a typical office building [2].It can be estimated from the fact, that at present, in United States, it takes 34 power plants, each capable of producing 500 MW of electricity, to power all the data centres in operation and by 2020, the nation will need another 17 similarly sized power plants to meet projected data centre energy demands as economic activity becomes increasingly digital[3]. More generally in a large data centre, the amount of electricity needed to run it is similar to that of a small town.
The major aspect to be understood in this context is the main consumers of energy

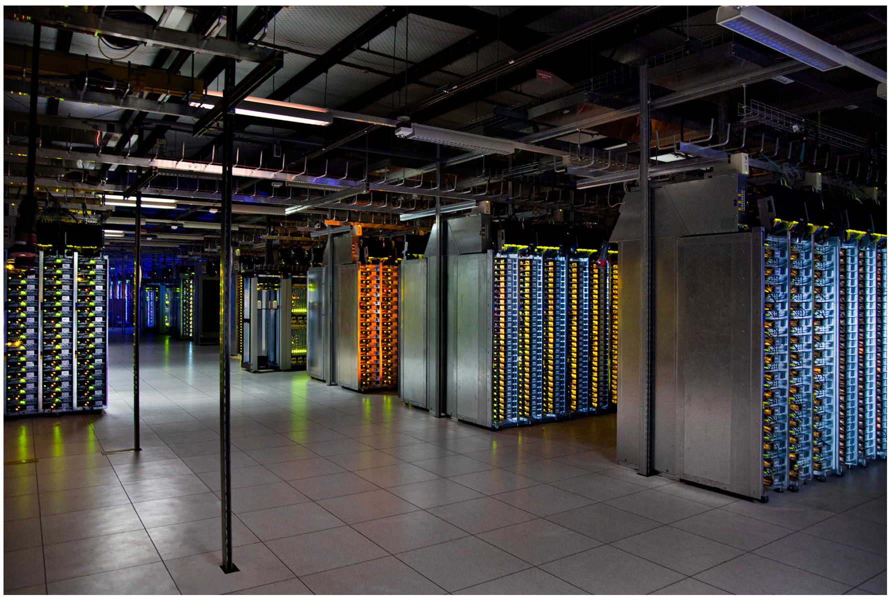


Figure 1.1: An Internal View of Google's Data Centre

in a Data Centre.Figure 1.2 refers to the typical energy consumption in a data centre.

There are two main sinkers of energy, firstly the power needed to run the servers and secondly the power needed to cool them down as the temperature increase caused by the high energy consumption can prove fatal to the life of servers and lead to inefficiencies in carrying out the commands that are to be executed by the servers. Another added burden is due to the need of redundancy of all these equipments to save the data centre from complete shutdown in case of some interruption, as the complete shutdown just for a short while can lead to the loss of billions of dollars.

Until recently , the computer science has been working rigorously to reduce the computational power consumption of the servers which forms a major percentage of the energy consumed and if kept under control could also reduce the power needed to cool down these servers , since less energy means less power and hence less heat dissipation. In this regard, the most significant technique which has been developed till date is known as Dynamic Voltage Frequency Scaling (DVFS). It is an accepted technique to lower the energy and power consumption of microprocessors which are the main users of computing power in data centres[4].The relation between the power consumed by the processor and the supply voltage is quadratic, hence lowering the supply voltage can reduce a significant amount of energy while lowering only the operating frequency can reduce the power consumption but the

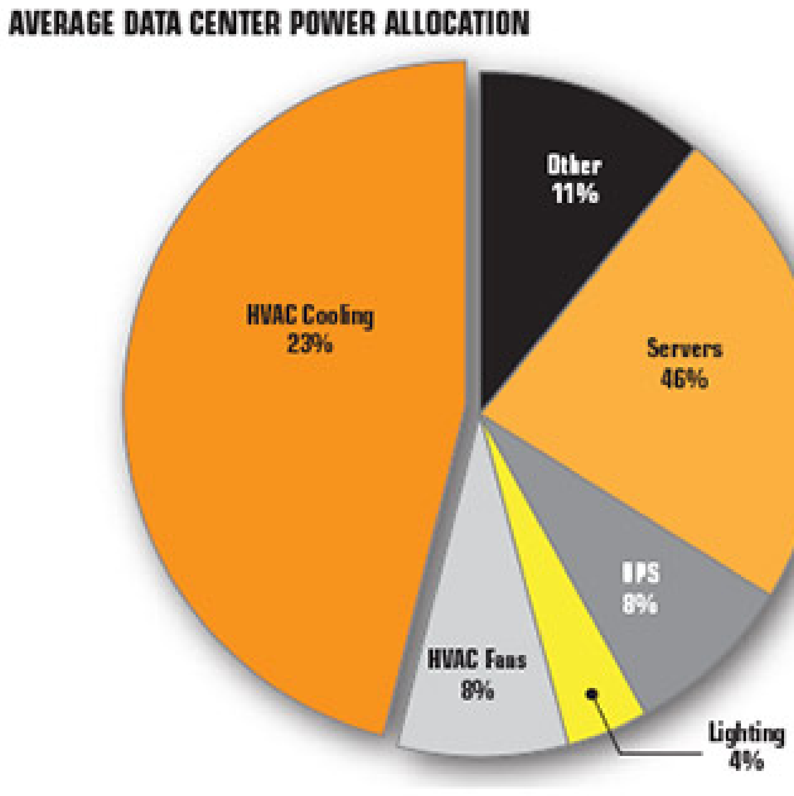


Figure 1.2: Typical Energy Consumption in a Data Centre

energy consumption remains the same because the computation needs more time to finish. Lowering the supply voltage and operating frequency reduces the power and energy consumption further. Different techniques like voltage scaling hardware loop and hardware technique for shutting down unused hardware modules have been devised till date to significantly and efficiently use DVFS to decrease consumption and increase efficiency [5, 6].

Having said that, up to date minimal efforts have been put into offering solutions from the point of view of system modelling and amalgamating them with classical control techniques for the optimization of energy and temperature control at local as well as central levels.Hence, in the thesis, the main objective is to develop the model of the server which will take into account the computational as well as thermal aspect of modelling and will be an amalgamation of the two. The equations used to model the server are relevant to duplicate their computing behaviour such as number of instructions to be executed, the energy consumed to execute these instructions and DVFS as well as the thermal behaviour which includes the dependence of temperature on the power consumed which in turn depends on the DVFS and the set of instructions to be executed.

The main advantage of this server modelling is that it is not very demanding computationally but at the same time gives complete and fast simulated results which can be very useful when designing the entire data centres and duplicate the detailed behaviour of the data centre quite fast and convincingly.

The temperature control of the server is implemented with the help of a discrete event controller and LQR which was successfully achieved having DVFS as input and temperature of the server as output. Furthermore, the modelling of racks along with both hot and cold aisles were successfully developed and then combined to form a small virtual data centre using Modelica as a software tool.

1.2 State of the Art

From the past many years, data centres have been the focus of extremely serious research as they are one of the major consumers of energy today and extreme need of energy optimization in today's world which is drowned in energy crisis is inevitable. With reference to this, different methodologies have been presented mostly based on dynamic voltage frequency scaling to reduce power and energy consumed by the processor. In [7], a proposed control loop of DVFS technique has been introduced.

The proposed dynamic voltage frequency scaling (DVFS) loop varies or sets the supply voltage Vdd and operating frequency according to the desired frequency which is predicted via the operating system and speed control circuit. The DVFS proposed loop has a high performance due to accuracy in progress, and can significantly improve processor energy efficiency. Another way to reduce energy consumption in data centres is by the reduction of the idle power of their servers [8] which consumes 60 percent of their peak power draw. Idle power refers to the electric power consumed by electronic and electrical appliances while they are switched off (but are designed to draw some power) or in a standby mode.

The method presented is called PowerNap, an energy-conservation approach where the entire system transitions rapidly between a high-performance active state and a near-zero power idle state in response to instantaneous load. Rather than requiring fine-grained power-performance states and complex load-proportional operation from each system component, PowerNap instead calls for minimizing idle power and transition time, which are simpler optimization goals. Based on the PowerNap concept, requirements and outlined mechanisms are developed to eliminate idle power waste in enterprise blade servers.

Because PowerNap operates in low efficiency regions of current blade centre power supplies, the Redundant Array for Inexpensive Load Sharing (RAILS) are introduced, a power provisioning approach that provides high conversion efficiency across the entire range of PowerNap's power demands. Using utilization traces collected from enterprise-scale commercial deployments, it is demonstrated that together, PowerNap and RAILS reduce average server power consumption by 74 percent.

Another technique proposed is based on the efficient resource management policy for virtualized Cloud Data centres[9]. The objective is to continuously consolidate these Virtual Machines leveraging live migration and switch off idle nodes to minimize power consumption, while providing required Quality of Service. Evaluation results show that the dynamic reallocation of Virtual Machines brings substantial energy savings. In [10], Pack and Cap is proposed , a novel technique for maximizing the performance of multi threaded workloads on multi-core processors within an arbitrary power cap.

Thread packing is introduced as a control knob that can be used in conjunction with DVFS to manage the power-performance tradeoff. It expands the range of feasible power caps, and enables fine-grained dynamic control of power consumption. In devising a multinomial logistic regression (MLR) classifier approach to identifying optimal operating points, possibility to automatically select Pareto-optimal DVFS and thread packing combinations during runtime is demonstrated successfully. Using a large body of characterization data gathered from the Princeton Application Repository for Shared-Memory Computers (PARSEC) benchmark suite, sophisticated classifier models are trained that encapsulate the workload dependence of the power-delay Pareto frontier. By performing model learning offline and exposing the models via lookup tables, the runtime overhead of the control scheme was reduced to a low-cost probability calculation.

Implementation of Pack and Cap on a real quad-core based system with a wide range multi-threaded workloads demonstrates that the method is capable of adhering to a power cap 82 percent of the time while maximizing performance, even in the absence of a power measuring device. Thread packing increases the range of feasible power constraints by an average of 21 percent when compared to DVFS alone and reduces workload energy consumption by an average of 51.6 percent compared to existing control techniques that achieve the same power range. In [11], DVFS along with per-core power gating (PCPG)as an additional power management knob for multi-core processors is used for processor power management.PCPG is the ability to cut the voltage supply to selected cores, thus reducing to almost zero the leakage power for the gated cores. Using a testbed based on a commercial 4-core chip and a set of real-world application traces from enterprise environments, it is shown that PCPG can significantly reduce a processor's energy consumption (up to 40 percent) without significant performance overheads. When compared to DVFS, PCPG is highly effective saving up to 30 percent more energy than DVFS. When DVFS and PCPG operate together they can save up to almost 60 percent.

Carrying on, [12] enables power-efficient management of enterprise workloads by exploiting a fundamental characteristic of data centres:"platform heterogeneity".This heterogeneity stems from the architectural and management-capability variations of the underlying platforms. An intelligent workload allocation method is defined that leverages heterogeneity characteristics and efficiently maps workloads to the best fitting platforms, significantly improving the power efficiency of the whole data centre.This allocation is performed by employing a novel analytical prediction layer that accurately predicts workload power/performance across different platform architectures and power management capabilities and achieves on average 20 percent improvements in power efficiency for representative heterogeneous data centre configurations, highlighting the significant potential of heterogeneity-aware management.

In [13], formal models for precedence-constrained parallel tasks, DVFS enabled clusters, and energy consumption are presented and aims to reduce energy consumption for high end computing. It studies the slack time for non-critical jobs, extends their execution time and reduces the energy consumption without increasing the task's execution time as a whole. By increasing task execution time within an affordable limit, this paper develops scheduling heuristics to reduce energy consumption of a tasks execution and discusses the relationship between energy consumption and task execution time. Models and scheduling heuristics are examined with a simulation study.

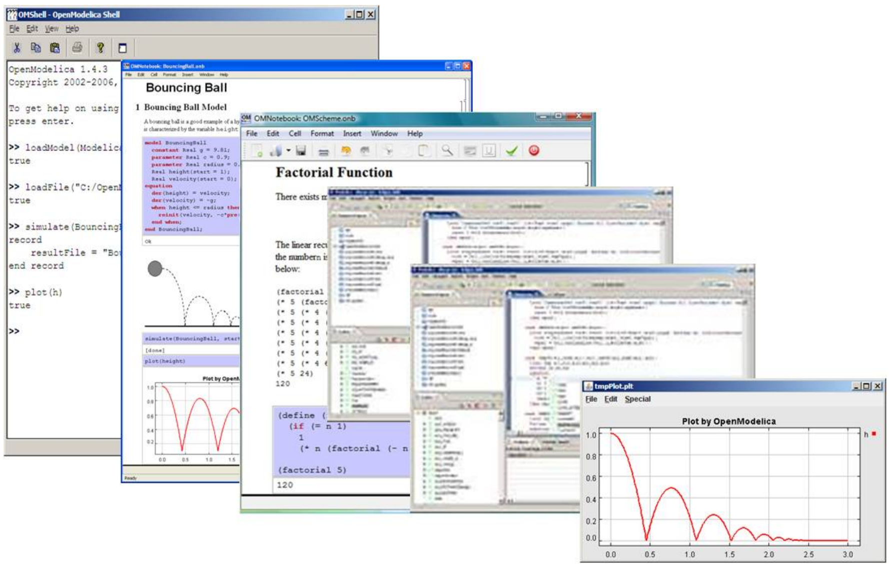
On the other hand,[14] provides an assessment of the current thermal modelling methodologies for data centres , with focus on the use of computational fluid dynamics (CFD) and heat transfer as analysis tools, and model validation while [15] focuses on importance of the HVAC (heating, ventilation, and air conditioning) methods reduce power consumption in data centres. Again in [16], two hierarchical thermal-aware power optimization techniques for data centres are proposed that are complementary to each other and achieve firsty lower overall system power with no performance penalty or secondly higher performance within the same power budget.

At the data centre level, trade off is the facility Heating, Ventilation and Air Conditioning (HVAC) power with server fan power by choosing between two thermal setpoints for the HVAC chiller based on the cooling zone utilization levels. This optimization can reduce total data centre total power by as much as 12.4 percent to 17 percent, with no performance penalty. At the server level, the trade off fan power and circuit leakage power by dynamically adjusting the server thermal setpoint, allowing the system to heat up when this saves more fan power than it costs in terms of leakage power. Evaluation shows that it reduces total server power by up to 5.4 percent with no performance penalty for workloads that heavily exercise a server.

The methodologies presented in the literature clearly indicate that up till now the entire focus has been either on developing classical DVFS techniques combined with resource allocation, load balancing and other power management techniques to reduce the computing power or on the other hand HVAC , CFD techniques to reduce the thermal power. There has been hardly any focus on developing a model of the server itself which takes into account the thermal as well as the computational aspects of the system, while at the same time being simple enough to be usable in data centre simulators that may contain hundreds or even thousands of servers. Hence the project carried out in this dissertation, focuses on the development of this simple server model which provides an almost complete behaviour from all the important aspects as well and when used in the context of the data centre, can be computationally very light as well as complete.

1.3 Modelica

The modelling language used to implement the models presented and discussed in this thesis is Modelica and a short introduction about this was deemed necessary to bring about totality in the documentation. Modelica is a language which is based on object oriented modelling. It is used for modelling of complex systems containing mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented sub components. Unlike other languages, it is developed by the non profit Modelica Association and is free, which was a big motivation behind its implementation in the thesis.While Modelica resembles object-oriented programming languages, such as C++ or Java, it differs in two important respects. First, Modelica is a modelling language rather than a conventional programming language. Modelica classes are not compiled in the usual sense, but they are translated into objects which are then exercised by a simulation engine. The simulation



```
OpenModelica 1.4.3
Copyright 2002-2006,

To get help on using
press enter.

>> loadModel(Modelic
true

>> loadFile("C:/Open
true

>> simulate(Bouncing
record
    resultFile = "Bo
end record

>> plot(h)
true

>>
```

Bouncing Ball

1 Bouncing Ball Model

A bouncing ball is a good example of a hy
is characterized by the variable height

```
model BouncingBall
  constant Real g = 9.81;
  parameter Real c = 0.9;
  parameter Real radius = 0.
  Real height(start = 1);
  Real velocity(start = 0);
equation
  der(height) = velocity;
  der(velocity) = -g;
  when height <= radius ther
    reinit(velocity, -c*pre
  end when;
end BouncingBall;
```

```
Ok
simulate(BouncingBall, start
[done]
plot(height)
```

Factorial Function

There exists m

The linear recu
the numbern is
below:

```
(factorial
(* 5 (facto
(* 5 (* 4 (
(* 5 (* 4 (
(* 5 (* 4 (
(* 5 (* 4 (
(* 5 (* 4 (
(* 5 (* 4 6
(* 5 24)
120
```

```
(define (
  (if (= n 1)
    1
    (* n (factorial (- n

(factorial 5)
120
```

Figure 1.3: Introduction to Modelica

engine is not specified by the language, although certain required capabilities are outlined. Second, although classes may contain algorithmic components similar to statements or blocks in programming languages, their primary content is a set of equations.
Modelica traces its origin to September 1996 by Hilding Elmqvist. The goal was to develop an object-oriented language for modelling of technical systems in order to reuse and exchange dynamic system models in a standardized format. Modelica 1.0 is based on the PhD thesis [17] of Hilding Elmqvist and on the experience with the modelling languages Allan,Dymola, NMF ObjectMath, Omola, SIDOPS+,and Smile. Hilding Elmqvist is the key architect of Modelica, but many other people have contributed as well. In September 1997, version 1.0 of the Modelica specification was released which was the basis for a prototype implementation within the commercial Dymola software system. In year 2000, the non-profit Modelica Association was formed to manage the continually evolving Modelica language and the development of the free Modelica Standard Library. In the same year, the usage of Modelica in industrial applications started.

1.4 Purpose and Organization of the thesis

In [19], like in the ancestor [1] of this dissertation, the server for the purpose of data centre energy optimization has been just modeled by its power consumption which has been approximated as linear in utilization between a fixed idle power and peak load power.

Similarly in [7] the modelling of the server is defined again by its power which is described as a combination of its dynamic, static and short circuited power, where the server's dynamic power represents the main portion of its power dissipation and is dependent on the supply voltage , the clock frequency and the collective switching capacitance. Again in [8], the server is approximated by the average computing power.

On the other hand, in [14] computational fluid dynamics (CFD) and heat transfer techniques have been used. Hence as can be concluded from the literature, all the server models present till date are either based on computational aspects of modelling or their thermal aspects and until now there has been no such model which combines the two to give a complete model.

The main purpose of this thesis is to provide a model of the server which provides a complete analysis – as long as a system-level attitude is taken – of its thermal as well as computational aspects while also highlighting the dependence of one on the other and then use this model to simulate the behaviour of the entire data centre. The main benefit of this model is that it is simple and complete never used before amalgamation of the two main phenomena, which are the main consumers of power in data centres but at the same time is not computationally heavy as in the case of for example CFD models, the simulations of which can last 24 hours or more and need extremely powerful processors. This model of the server, when fitted into the context of data centres provide computationally less demanding yet detailed simulated behaviour which can be extremely useful for the purpose of their energy optimization.

The following chapters of the thesis are so structured:

- Chapter 2 provides an explanation of the basic thermodyamic equations and concepts used for the modelling of elementary blocks of the overall data centre.

- Chapter 3 discusses the implementation, simulation and results obtained when the concepts discussed in Chapter 2 are simulated in Modelica

- Chapter 4 provides an overview of different controllers used to govern the server thermal behaviour, thereby assessing the model as suitable for the synthesis of such local controllers, and illustrates their successful implementation in Modelica.

- Chapter 5 concludes with the objectives fulfilled, and sketches out future works in this direction.

Chapter 2

Modelling Bricks of Data Centres

2.1 Introduction

This chapter introduces the basic concepts and equations which have been used to simulate the behaviour of the data centre starting from servers which form the basic entities in a data centre to the air flowing through the hot and cold aisles. Then these servers are arranged in racks, which are essential for the proper working and design of a data centre. All these components have been modelled using the basic concepts of thermodynamics and then combined to form a working model of the data centre.

2.2 Modelling of the Server

This is the most important section of the thesis. As discussed in the first chapter, this section presents a novel method to model the server. The need for this modelling arises from the fact that all the previous models which have been used to represent the server are either in some cases too simple to present a credible model of the server in terms of its power and performance or so computationally heavy and time consuming that the simulation needs days to complete. On the other hand, these models fail to describe the relationship between the thermal power and the computational load of the server which is extremely important for understanding how both of these aspects together effect the performance of the server, which is the most important concern in the computer science world.

In this aspect, it is very important to understand that the thermal power of the server ultimately depends on its computational load, be it the input rate of instructions which it needs to execute or the kind of instructions it executes, since different instructions require different levels of energy with some tasks requiring more energy than the others. Hence with this concept in mind, our model in a very simple yet credible way relates the thermal power of the server to its computational load. The model is represented by very simple and fine grained set of equations, its main aim being firstly to present the system as continuous. Secondly, this model when used in the context of a data centre where a large number of servers need to be simulated can provide affordable complexity.

Starting with the short description of the server, the term “server” cannot be defined in a single way. A typical server is shown in figure 2.1 below.

Figure 2.1: Single server in a data centre

A server is a physical computer dedicated to running one or more such services (as a host), to serve the needs of the users of the other computers on the network. Depending on the computing service that it offers, it could be a database server, file server, mail server, print server, web server or others. Servers provide essential services across a network, either to private users inside a large organization or to public users via the Internet. In hardware sense, the word server typically designates computer models intended to hosting software applications under the heavy demand of a network environment. In this client-server configuration, one or more machines, either a computer or a computer appliance, share information with each other, with one acting as a host for the others. While nearly any personal computer is capable of acting as a network server, a dedicated server will contain features making it more suitable for production environments. These features may include a faster CPU(Central Processing Unit), increased high performance RAM (Read Only Memory) and typically more than one large hard drive. For the purpose of this work, however , there is no need to model accurately the hardware part, since a simple model, yet accurate concerning thermal and computational power must be enough.

The server is described by two inputs and one output. The first input is

the input rate of instructions (roi) that the system needs to execute and for simplification , it is normalized with respect to the maximum rate of instructions that the server is capable to execute. The second input to the server is the dynamic voltage frequency scaling (DVFS) command which has been assumed to be already determined from a preexisting loop in the server and it ranges between 0 and 1.
This DVFS command decides the rate of instructions based on its scaling, which the server is actually able to execute given the power limitations and hence these rate of instructions are given by

$$qreq = Cmax * DVFS; \tag{2.1}$$

Once the rate of instructions from the server determined by the DVFS command is calculated (qreq), the net rate of instruction queue (dn/dt) which are actually executed by the server is given by the input rate of instructions which are requested to be executed by the server (roi) minus the real rate of instructions which the server is able to execute.

The rate of instructions length executed by the server is given by

$$\frac{dn}{dt} = Cmax * roi - qreq \tag{2.2}$$

where
n >=0
Cmax=Maximum rate of server capacity to execute instructions (instructions/second)
DVFS=Dynamic Voltage Frequency Scaling[0,1]
roi=normalized input rate of instructions to execute w.r.to Cmax
qreq=requested rate of execution (instructions/second)
Another limitation comes in the form of instruction queue (n) which can never be less than zero and further limits the rate of instructions which the server is able of execute (qact) and is given

$$qact = Cmax * roi - \frac{dn}{dt} \tag{2.3}$$

where
qact=actual execution rate which the server can execute keeping the constraint n >=0 under check

So equation 2.3 gives the actual computational load which the server executes and based on this load , the thermal power can be computed. In this case , for simplicity the equation to compute the power is based on the assumption that each instruction uses the same energy which is not the actual case and in real cases energy needed depends on the different tasks which need to be executed.

$$P = qact * epi \tag{2.4}$$

where

P=power consumed by the server when executing instructions (Watts)

epi=energy per instruction (Pico joules)

Finally, the model of the server has one output , which is the temperature of the server. This temperature depends on the thermal power needed to execute the computational load of the server and the exchange of heat with the external environment through a heat port and is presented through the thermal phenomena in equation 2.5.

$$C * \frac{dTcpu}{dt} = P + QFlow \tag{2.5}$$

where

C= Heat Capacity (Joules/Kelvin)

Tcpu = Temperature of the server depending on the power consumed by the executions of the instructions in the server and heat flow between the external environment and the server(Kelvin).

QFlow =heat flow rate between the server and the external environment(Watts)

Hence this modelling of the server provides an understanding of accessing thermal behaviour together with the computational load, which provides an important break through in this field.

2.3 Modelling of the Air Compartment

The modelling of the air requires the knowledge and understanding of a large number of thermodynamic and transport properties related to it. A general process of the air flow in the atmosphere is depicted in the figure 2.2

The air is a heterogeneous mixture of gases and liquid and solid particles in suspension (clouds, dust, microorganisms...). As is a well known fact, all natural air is humid; dry air must be artificially obtained. In many engineering problems,

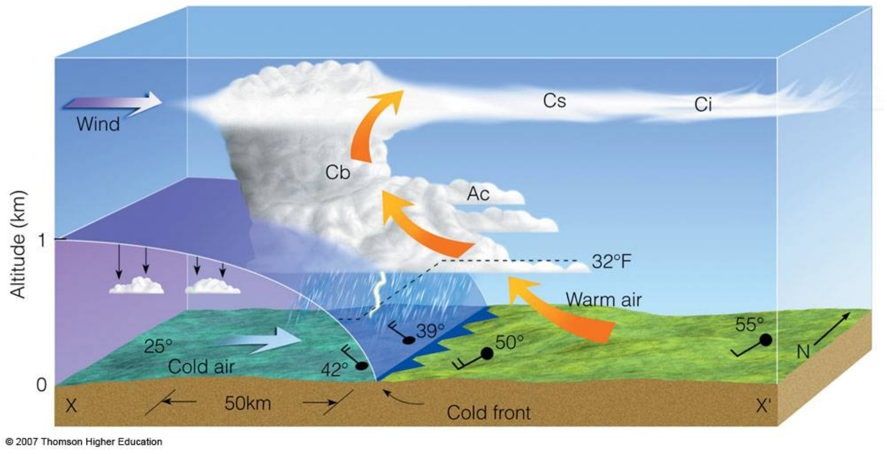


Figure 2.2: Air Flow in the Atmosphere

where there is little or no change in composition of the air, humid air formulation is not needed and air can be treated as a pure substance (from aircraft lifting to most heat transfer problems). But in some cases, the change in composition of the air may be crucial, either because condensation occurs or because air entrains water from some source, as from vital meteorological processes, to artificial air conditioning and evaporative cooling. Hence, the modelling of moist air is of relevance to our project.

2.3.1 Modelling of the Moist Air

Humid air is the most important gaseous mixture in science and engineering: from breathing to meteorology and air conditioning and can be modeled as a binary gas mixture of dry air and water vapour because none of the components of dry air is highly soluble in liquid water, and dry-air composition can be considered invariable. For general information, a binary system is a particular case of the more general multi component system in which only two components are present. Th term mixture refers to the fact that both substances to be treated are of equal footing.

The two-phase binary mixture of water and air is not an ideal mixture in both phases, but for the gaseous phase the ideal mixture model is very appropriate, and Raoult's law for the two-phase equilibrium of water is still valid. It is one of the most important law of thermodynamics and was established by French physicist François-Marie Raoult in 1882. It states that the partial vapor pressure of each component of an ideal mixture of liquids is equal to the vapor pressure of the pure component multiplied by its mole fraction in the mixture and is used in the modelling equation 2.7. The thermodynamic state of humid air, as a binary homogeneous mixture (of dry air and water vapor, in gas phase), requires three intensive variables to be fully specified: pressure, temperature and humidity, the latter being determined by several different variables, all of them related. Given as input initial temperature, pressure and absolute humidity , the terms and equations which define the thermodynamic state of moist air can be written as:

The vapour mass fraction of the total air is given as

$$x = \frac{X}{1+X} \tag{2.6}$$

where

X=input prescribed absolute humidity of dry air(kg/m^3)

Applying Raoult's law to the mixture, the partial pressure of the water vapor is given by

$$pv = p * \frac{X}{X+0.622} \tag{2.7}$$

where

pv=vapour partial pressure

and

p=input prescribed pressure(Pascals)

$$pvs = 610.10 * e^{\frac{T-273.15}{T-273.15+238.3}} * 17.2694 \tag{2.8}$$

where

pvs=saturated vapour partial pressure

and

T=input prescribed temperature(Kelvins)

The relative humidity marks the proximity of the saturation and is an important parameter in describing the great effects of condensation and evaporation. The amount of water vapor in the air at any given time is usually less than that required to saturate the air. The relative humidity is the percent of saturation humidity, generally calculated in relation to saturated vapor density and is given by

$$phi = \frac{pv}{pvs} \tag{2.9}$$

where

phi=relative humidity

In case of saturation equals 1

But relative humidity does not give an indication of how much vapour there actually is (unless complemented by p-T-values), as specified by pv.The humidity ratio or absolute humidity, X, related the mass of the dry air with that of the mass of water vapour. The reason for not using the total mass but the mass of dry air is that air and water are very dissimilar substances, and for the range of temperatures and pressures envisaged, one of the components (dry air) may be thought as permanently in the gas phase, while water vapour can be easily changed, and thus it is advantageous to refer concentrations, enthalpies, and other thermodynamic functions, to the conservative mass of dry air.

$$Mv = Ma * X \tag{2.10}$$

where

Ma=Total Dry Air Mass

Mv= Total Vapour Mass

The wet bulb temperature is the temperature a small wet-object would reach, by evaporative cooling, when exposed to an air flow. When the combined heat and mass transfer problem is solved, it happens that the value of this temperature is approximately the adiabatic saturation temperature. Because it is easy to measure (just blowing over a thermometer with its bulb surrounded by a small mesh soaked in water), it was customarily used to measure humidity by rotating a set-up with two equal mercury thermometers, one of them with the bulb wrapped with a wick soaked in water (sling psychrometer).

$$Twb = 273.15 - 238.3 + \frac{T - 273.15}{17.2694 * ln(\frac{pv}{610.78})} \tag{2.11}$$

where

Twb=Temperature of the wet bulb

$$Xs = 0.622 * \frac{pvs}{p - pvs} \tag{2.12}$$

where

Xs=Absolute humidity of dry air at saturation

Another important characteristic, which is extremely important for the modelling of moist air is enthalphy. The word enthalpy is based on the Greek enthalpein , which means "to warm in". Enthalpy is the amount of heat content used or released in a system at constant pressure. Enthalpy is usually expressed as the change in enthalpy. The change in enthalpy is related to a change in internal energy (U) and a change in the volume (V), which is multiplied by the constant pressure of the system. Hence the Enthalphy(H) is the sum of internal energy (U) and the product of pressure and volume (PV) given by the equation:

$$H = U + PV \tag{2.13}$$

If temperature and pressure remain constant through the process and the work is limited to pressure-volume work, then the enthalpy change is given by the equation:

$$\Delta H = \Delta U + P\Delta V \tag{2.14}$$

Also at constant pressure the heat flow (q) for the process is equal to the change in enthalpy defined by the equation:

$$\Delta H = q \tag{2.15}$$

When the temperature increases, the amount of molecular interactions also increases. When the number of interactions increase, then the internal energy of the system rises. According to the equation 2.14, if the internal energy (U) increases then the ΔH increases as temperature rises. The equation for heat capacity and equation 2.15 can be used to derive this relationship.

$$C = \frac{q}{\Delta T} \tag{2.16}$$

Under constant pressure, substituting equation 2.15 into 2.16

$$C_p = \left(\frac{\Delta H}{\Delta T}\right)_P \tag{2.17}$$

Under the assumption of constant pressure throughout, the enthalpy relating thermodynamics of our case is defined as

$$ha = cp_a * (T - 273.15) \tag{2.18}$$

where

ha=specific enthalpy of dry air(J/Kg)

and

cp_a=Specific heat capacity of dry air

$$hv = h_v 3pt + cp_v * (T - 273.15) \tag{2.19}$$

where

hv=specific enthalpy of water vapour(J/Kg)

cp_v=Specific heat capacity of water vapour

and

h_v3pt=Specific enthalpy of water vapour at triple point [J/Kg]

The triple point of a substance is the temperature and pressure at which the three phases (gas, liquid, and solid) of that substance coexist in thermodynamic equilibrium.

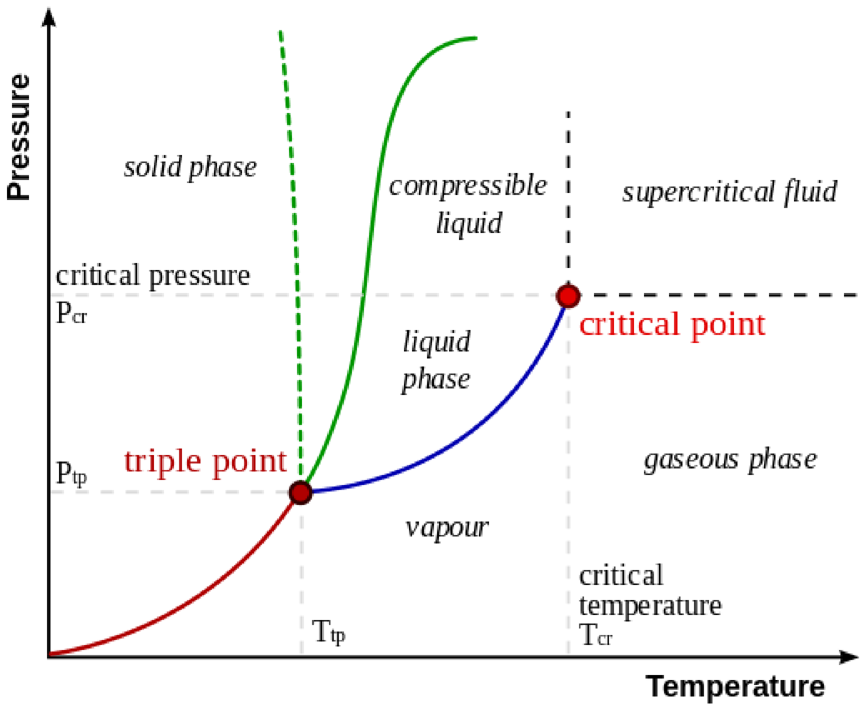


Figure 2.3: Triple point

$$h = ha + X * hv \tag{2.20}$$

where

h=specific enthalpy of moist air(J/Kg)

$$hl = cp_l * (T - 273.15) \tag{2.21}$$

where

h=specific enthalpy of liquid water(J/Kg)
and
cp_l=Specific heat capacity of liquid water

The density of air varies as the temperature and moisture content in the air varies. When the temperature increases, the higher molecular motion results in an expansion of the volume and thus decreasing the density. The density of a gas, either it is dry air, water vapor or a mixture of dry air and water vapor - moist or humid air, can be calculated on basis of the Ideal Gas Law.

The Ideal Gas Law can be expressed as

$$p * V = m * R_x * T \tag{2.22}$$

where p = *absolute pressure (N/m^2)*
V = *volume of gas (m3)*
m = *mass of gas (kg)*
R_x = *individual gas constant (J/kg.K)*
T = *absolute temperature (Kelvin)*

The density can be expressed as

$$d = \frac{m}{V} \tag{2.23}$$

and equation 2.22 becomes

$$p = d * R * T \tag{2.24}$$

The individual gas constant can be expressed with the universal gas constant and the molecular weight of a gas as

$$R_x = \frac{Ru}{M_g} \tag{2.25}$$

where
M_g = *molecular weight of the particular gas*
and
R_u = *8314.47 = universal gas constant (J/(kmol K))*

Hence to define the density of moist air,equation 2.26 based on the Ideal Gas Law is used.

$$d = \frac{(p - pv)}{(Ra * T)} + \frac{pv}{Rv * T} \tag{2.26}$$

where

d=density of moist air (Kg/m^3)

Ra=Gas constant of dry air

and

Rv=Gas constant of water vapour in moist air

$$cp = cp_a + X * cp_v \tag{2.27}$$

cp=moist air specific heat capacity(J/Kg)

$$R = Ra + Rv * X \tag{2.28}$$

where

R=equivalent gas constant of moist air

2.3.2 Balance Equations

All the balance equations, whether they involve the balance of mass in a system or the balance of energy are based on the very basic principle of physics, which is the conservation principle. It states that matter can neither be created nor destroyed but can be changed from one form to another. In our case, two balances that are used to duplicate the behaviour of air compartment, are mass balance and energy balance. The use of energy balance in the modelling holds relevance , since the temperature is not constant and a lot of heat is generated due to the power consumed by the servers of the data centre.

The total mass of the moist air is given by

$$Ma + Mv = V * d \tag{2.29}$$

where

V=Volume

$$ea = \frac{(h - p)}{d} \tag{2.30}$$

where

ea=Specific energy of the moist air

$$Ea = \frac{(Ma + Mv)}{ea} \tag{2.31}$$

where

Ea=Energy of the moist air

The mass balance is described by total continuity equation which states that the time rate of change of mass inside any system must be equal to the mass entering into the system minus the mass exiting the system.

$$wtin = wain * (1 + X) \tag{2.32}$$

where

wtin = Total mass flow rate at the input

and

wain=Mass flow rate of the dry air at the input

$$wtout = waout * (1 + X) \tag{2.33}$$

where

wtout = Total mass flow rate at the output

and

waout=Mass flow rate of the dry air at the output The total mass balance is given by

$$\frac{d(Ma + Mv)}{dt} = wtin + wtout - wc \tag{2.34}$$

where

wc = Condensed water mass flow rate The mass balance of the water vapour is given by

$$\frac{dMv}{dt} = wain * xain + waout * xaout - wc \tag{2.35}$$

where

xain = Absolute humidity of the dry air at the input

and

xaout = Absolute humidity of the dry air at the output

Since, there is a change of temperature and consumption of power resulting in heat generation, so in order to model our system correctly, energy balance is extremely vital for the purpose.

$$Cw * \frac{dTw}{dt} = Qawsen + Qawlat + QFlow_{wall} \tag{2.36}$$

where

Cw = Heat Capacity of the Wall(Joules/Kelvin)

Tw = Wall Temperature (Kelvin)
Qawsen=sensible heat
Qawlat=Latent Heat
and
$QFlow_{wall}$ *=Heat Flow Rate between wall and the environment(Watt)*

Latent and sensible heat are types of energy released or absorbed in the system. Latent heat is related to changes in phase between liquids, gases, and solids. Latent heat is the energy absorbed by or released from a substance during a phase change from a gas to a liquid or a solid or vice versa. If a substance is changing from a solid to a liquid, for example, the substance needs to absorb energy from the surrounding environment in order to spread out the molecules into a larger, more fluid volume. If the substance is changing from something with lower density, like a gas, to a phase with higher density like a liquid, the substance gives off energy as the molecules come closer together and lose energy from motion and vibration. For example, when water is boiled over a stove, energy is absorbed from the heating element and goes into expanding the water molecules into a gas, known as water vapor. When liquid water is put into ice cube trays and placed in the freezer, the water gives off energy as the water becomes solid ice. This energy is removed by the freezer system to keep the freezer cold. Hence the definition itself is self explanatory to explain its dependence on condensed water flow rate and the enthaphies of the different phases as is described in equation 2.37

$$Qawlat = wc * (hv - hl) \tag{2.37}$$

where
Gaw=Air− Wall Thermal Conductance

Sensible heat is related to changes in temperature of a gas or object with no change in phase and is given by the equation 2.38.

$$Qawsen = Gaw * (T - Tw) \tag{2.38}$$

Finally the rate the change of the output temperature is given by equation 2.39

$$d * V * cp * \frac{dTout}{dt} = wc * cp * (T - Tout) + QFlow_{air} \tag{2.39}$$

where
Tout = Output Temperature (Kelvin)
and
$QFlow_{air}$ *=Heat Flow Rate between input and output air(Watt)*

2.4 Modelling of the Rack and Data Centre

Once the basic blocks have been modeled, they need to be joined together into racks. The rack contains multiple mounting slots called bays, each designed to hold a hardware unit secured in place with screws. A single rack can contain multiple servers stacked one above the other, consolidating network resources and minimizing the required floor space. The rack server configuration also simplifies cabling among network components. In an equipment rack filled with servers, a special cooling system is necessary to prevent excessive heat buildup that would otherwise occur when many power-dissipating components are confined in a small space.If servers are logically placed in rows with the front of the racks (and servers) all facing the same direction, then a consistent airflow direction throughout the rows of racks is achieved. However, if several parallel rows of racks are placed with the same orientation, a significant efficiency problem arises. The hot exhaust air from the first row of racks gets sucked into the "cool" air intakes of the second row of racks. With each progressive row, the air temperature increases as hot air is passed from one row of servers to the next.To overcome this problem, the rows of server racks should be oriented so that the fronts of the servers face each other. In addition, the backs of the server racks should also face each other. This orientation creates alternating "hot aisle/cold aisle" rows.
Hot aisle/cold aisle is a layout design for server racks and other computing equipment in a data centre. The goal of a hot aisle/cold aisle configuration is to conserve energy and lower cooling costs by managing air flow. In its simplest form, hot aisle/cold aisle data centre design involves lining up server racks in alternating rows with cold air intakes facing one way and hot air exhausts facing the other. The rows composed of rack fronts are called cold aisles. Typically, cold aisles face air conditioner output ducts. The rows the heated exhausts pour into are called hot aisles. Typically, hot aisles face air conditioner return ducts. A simple data centre with multiple racks is shown in Figure 2.4.
As can be seen in Figure 2.4, all the servers inside the rack are arranged in such a way that heat generating sides of all the servers of the rack are on one side and this side is associated with the hot aisle and is the hotter section of the data centre with hot air (represented by red arrows in Figure 2.4). The cold aisle is formed by the cold air coming from the air conditioner and this cold air (represented with blue arrows in Figure 2.4) is send to the non heat generating side of the rack. Then also there is re-circulation of this hot air inside the data centre and results in overall increase of the temperature inside the data centre room. The hot and cold aisles in our system have been modeled with the help of air compartments

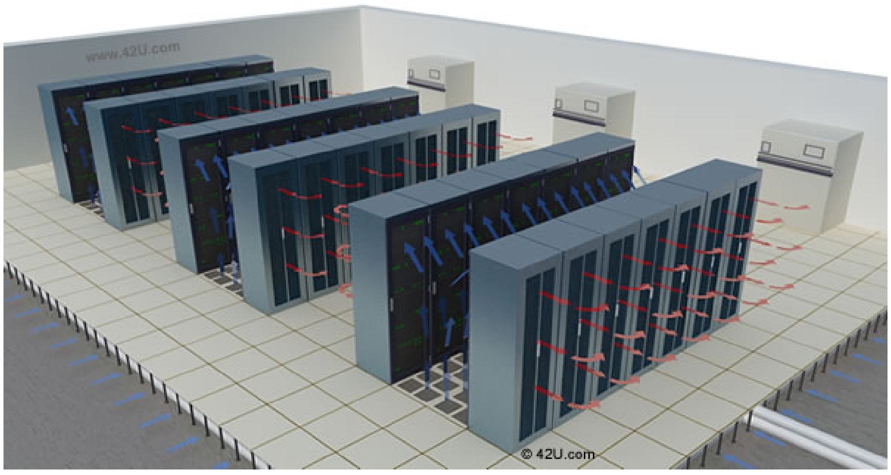


Figure 2.4: Racks arranged in a Data Centre

presented in section 2.2.

The issue with hot aisle/cold aisle designs is that the air is free to move wherever it will.A containment system can be used to isolate hot aisles and cold aisles from each other and prevent hot and cold air from mixing. Containment systems started out as physical barriers that simply separated the hot and cold aisles with vinyl plastic sheeting or Plexiglas covers. Today, vendors offer plenums and other commercial options that combine containment with variable fan drives (VFDs) to prevent cold air and hot air from mixing. A typical containment is shown in Figure 2.5.

With reference to our model of the server obtained in section 3.2, it is very important to mention here that the model which was designed for the purpose of being used in the context of data centre is extremely useful. The main advantage is that it provides a complete description of the thermal as well as the computational behaviour of the system. As is known, data centre contains servers ranging from hundreds to thousands and hence in order to simulate the complete model of the data centre can be demanding computationally. Our model of the server , when used in the context of a data centre firstly provides a very good description of thermal power consumed by the data centre together with the computational load and secondly and mostly importantly , the simulations carried out are not computationally demanding at all. This model can be very useful in the future to help develop new methods and technologies when working for the optimization of energy in these data centres.

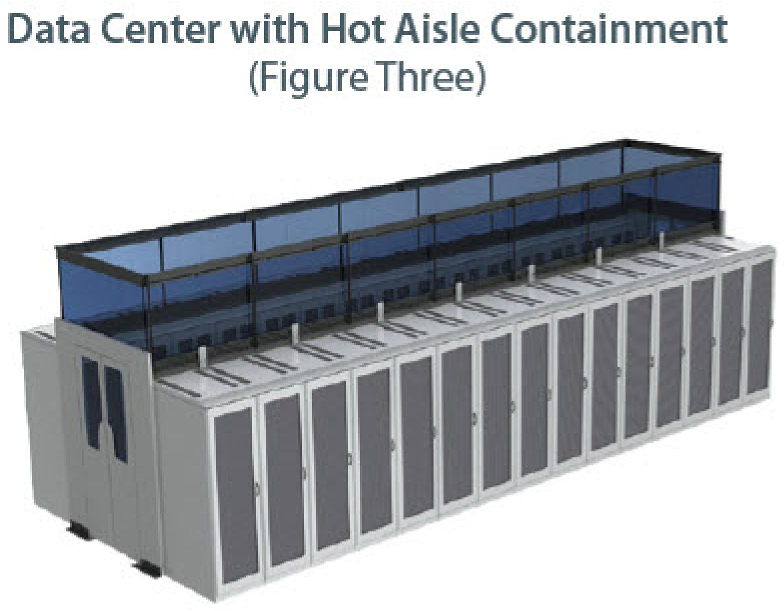


Figure 2.5: Data Centre with Hot Aisle Containment

2.5 Conclusion

This chapter has successfully presented all the basic concepts and equations which have been used to model our data centre. The next chapter will show the implementation of these concepts, amalgamation of these blocks to bring into existence a working model of the data centre and simulation results associated with it.

Chapter 3

Simulation and Implementation of Data Centre Model

3.1 Introduction

The previous chapter gave a detailed overview of the many concepts and equations used to model our server, air compartments and finally how these blocks can be combined to duplicate the behaviour of a data centre. This chapter presents an insight into obtained model by amalgamation of its basic blocks, its implementation in Modelica and the simulation results obtained.

3.2 Simulation and Implementation of Server

This section is divided into two subsections. The first section shows the implementation and results of the server alone while the second section shows the server modeled along with the air compartment.

3.2.1 Server without Air Compartment

This section presents a working model of the server. The equations used to describe this system have already been described in section 2.1 of Chapter 2.

Coding and Block Diagram Implementation in Modelica:

The code is presented in Code Snippet 3.1 below.

```
model rackmodel
  parameter SI.HeatCapacity Ca = 0.1 "Thermal cap";
  parameter SI.Temperature Tstart = 273.15 + 20 "Initial T";
  //SI.Temperature Tairin(start = Tstart) "Input Temperature";
  SI.Temperature Tair(start = Tstart) "Output Temperature";
  parameter Real rhoair = 1.2754 "kg/m^3";
  parameter Real G = 1;

  Modelica.Blocks.Interfaces.RealOutput Tout annotation(Placement(visible = true,
     transformation(origin = {100, 20}, extent = {{-10, -10}, {10, 10}}, rotation
     = 0), iconTransformation(origin = {85, 15}, extent = {{-15, -15}, {15, 15}},
     rotation = 0)));
  Modelica.Blocks.Interfaces.RealInput Temcpu annotation(Placement(visible = true
    , transformation(origin = {-100, -20}, extent = {{-10, -10}, {10, 10}},
    rotation = 0), iconTransformation(origin = {-85, -25}, extent = {{-15, -15},
    {15, 15}}, rotation = 0)));
  Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_a Heatportout annotation(
    Placement(visible = true, transformation(origin = {0, -100}, extent = {{-10,
    -10}, {10, 10}}, rotation = 0), iconTransformation(origin = {5.18249e-016, -
    80}, extent = {{-20, -20}, {20, 20}}, rotation = 0)));
  Modelica.Blocks.Interfaces.RealInput Tin annotation(Placement(visible = true,
    transformation(origin = {-85, 55}, extent = {{-15, -15}, {15, 15}}, rotation
    = 0), iconTransformation(origin = {-85, 55}, extent = {{-15, -15}, {15, 15}},
     rotation = 0)));

equation
  Ca * der(Tair) = rhoair * Heatportout.Q_flow * Ca * (Tin - Tair) +
    Heatportout.Q_flow;
  Heatportout.T = Tair;
  Tout = Tair;

end rackmodel;
```

Code Snippet 3.1: Modelica Code of a Data Centre Server

The simulation block implementation is demonstrated with the help of figure 3.1 ,which shows the server model connected to the inputs DVFS and roi and the heat port exchanging heat with the external environment through a thermal conductor. The output of this block is temperature which has a strong dependence on the consumed power, and proportionally increases with the load of the server and hence the server has a strong probability of trespassing its nominal operating point as the data centres have mostly very heavy set of instructions lined up which need to be fulfilled.

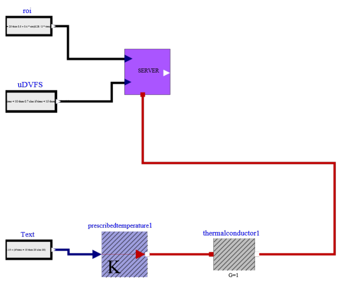


Figure 3.1: Block Diagram of Server in Modelica

Simulation and Results:

Figure 3.2 and 3.3 show the inputs given to the system in the form of input set of instructions and Dynamic Voltage Frequency Scaling.

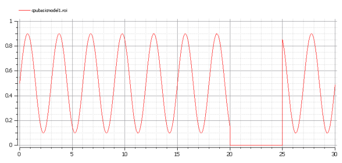


Figure 3.2: Normalized input set of Instructions (Roi)

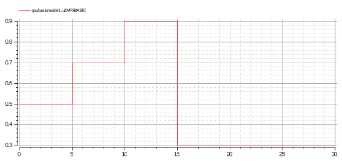


Figure 3.3: Dynamic Voltage Frequency Scaling

As can be seen in Figure 3.5 , the temperature increases and decreases as the input varies. For a high DVFS , the temperature and power (Figure 3.6) increases proportionally and vice versa. Also roi effects the temperature and power directly, as can be seen between 20 and 25 seconds, when roi is zero, temperature is the lowest.

Also, the quick drop in temperature and power at around 25 seconds can be attributed to the fact that at that point of time, there are no instructions to be executed, which is quite obvious from Figure 3.4.

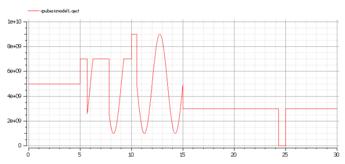


Figure 3.4: Actual rate of instructions that are being executed (qact)

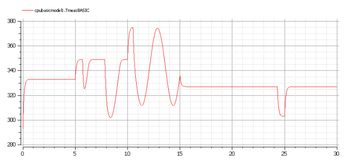


Figure 3.5: Output Temperature of the Server

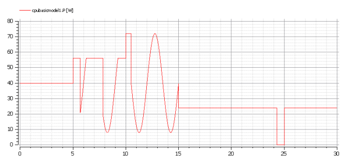


Figure 3.6: Consumed Power

3.2.2 Server With Air Compartment

The modelling of server should also involve the the phenomenon of heat exchange with its environment, hence the server needs to be modeled along with the air compartment, which describes the heat flow rate, enthaphy and mass flow rate and how they effect the server and the environment around it.

Coding and Block Diagram Implementation in Modelica:

```
model AirVolumeWithWall_Condensing
  extends Interfaces.AirPort.PartialTwoPort_a;
  EEB_source4hafsa.Media.Substances.MoistAir air;
  EEB_source4hafsa.Media.Substances.MoistAir wallsat;
  parameter SI.Volume V = 0.001 "volume";
  parameter SI.Pressure Pstart = 101325 "Initial moist air pressure";
  parameter SI.Temperature Tstart = 273.15 + 20 "Initial moist air temperature";
  parameter SI.MassFraction Xstart = 0.001 "Initial absolute umidity [kg_H20/kg_DA]";
  parameter SI.HeatCapacity Cw = 50 "wall heat capacity [J/K]";
  parameter SI.ThermalConductance Gaw = 100 "air-wall thermal conductance";
  parameter SI.ThermalConductance Gair = 1 "air thermal conductance";
  SI.Mass Ma "Total dry air mass";
  SI.Mass Mv "Total vapour mass";
  SI.Energy Ea "Energy of the moist air";
  SI.SpecificEnergy ea "Specific energy of the moist air";
  SI.Pressure P(start = Pstart) "Pressure of the air";
  SI.Temperature Ti "Temperature of the air";
  SI.Temperature Tw "Wall temperature";
  SI.Temperature Tout "Temperature of the output air";
  SI.MassFraction Xi "Water vapour mass fraction [kg_H20/kg_DA]";
  SI.MassFlowRate wt1 "Total mass flow rate at flange 1";
  SI.MassFlowRate wt2 "Total mass flow rate at flange 2";
  SI.MassFlowRate wc "Condensed water mass flow rate";
  SI.HeatFlowRate Qawsens "Sensible heat";
  SI.HeatFlowRate Qawlat "Latent heat";
  Interfaces.HeatPort.HeatPort heatPort annotation(Placement(transformation(
    extent = {{-60, 40}, {60, 60}}), iconTransformation(extent = {{-70, 40}, {70,
    60}})));
  EEB_source4hafsa.Interfaces.HeatPort.HeatPort heatPortair annotation(Placement(
    visible = true, transformation(origin = {-25, -75}, extent = {{-35, -15}, {
    35, 15}}, rotation = 0), iconTransformation(origin = {-5, -70}, extent = {{-
    15, -10}, {15, 10}}, rotation = 0)));
initial equation
  Xi = Xstart;
equation
  // Total mass balance
  wt1 = wa1 * (1 + Xi);
  wt2 = wa2 * (1 + Xi);
  der(Ma + Mv) = wt1 + wt2 - wc;
  der(Mv) = wa1 * actualStream(air_flange1.xa) + wa2 * actualStream(
    air_flange2.xa) - wc;
  Ma + Mv = V * air.d;
  // Mass fraction [kg_H20/kg_da]
  Mv = Ma * Xi;
  Xi = air.X;
  pa1 = pa2;
  P = pa1;
  air.p = P;
  // Energy balance in the air volume
  der(Ea) = wt1 * actualStream(air_flange1.ha) + wt2 * actualStream(
    air_flange2.ha) - Qawsens - Qawlat;
  // Energy of the moist air
  Ea = (Ma + Mv) * ea;
  ea = air.h - P / air.d;
  // specific energy of the moist air
  // Wallsat condition
  wallsat.p = P;
  wallsat.T = Tw;
  wallsat.phi = 1.0;
  // Condensed water
  wc = if air.X > 1e-006 and air.X > wallsat.X then Gaw / air.cp * (air.X -
    wallsat.X) else 0.0;
  // Thermal connector
  Ti = air.T;
  heatPort.T = Tw;
  heatPortair.T = Tout;
  // Heat transfer
  Cw * der(Tw) = Qawsens + Qawlat + heatPort.Q_flow;
  Qawsens = Gaw * (air.T - Tw);
  Qawlat = wc * (air.hv - wallsat.hl);
  // Heat transfer between input and output air
  air.d * V * air.cp * der(Tout) = Gair * (air.T - Tout) + heatPortair.Q_flow;
  // air.d * V * air.cp * der(Tout) = wc * air.cp * (air.T - Tout) +
    heatPortair.Q_flow;
  //air.d * V * air.cp * der(Tout) = heatPortair.Q_flow;
  // air humidity ratio boundary conditions
  xaout1 = Xi;
  xaout2 = Xi;
  // enthalpies boundary conditions
  haout1 = air.h;
  haout2 = air.h;

end AirVolumeWithWall_Condensing;
```

Code Snippet 3.2: Modelica Code of an Air Compartment with Saturated Wall

The modelling scenario represents a single server which is inside a room . There are two heat exchange processes, one between the room and the external environment through its wall and the second exchange is between the server and the environment of the room through the wall of the server. Three heat ports can be seen in Figure 3.7.

Two heat ports are associated with the modelling of the air compartment, one represents the transfer rate at which heat flows from the room to the external environment and vice versa. The second heat port represent the heat flow rate inside the room which is also connected to heat port of the server and represents the heat transfer between the server and the room and vice versa. Hence any change in the temperature of the server because of its computing power or any other energy consumption directs effects the temperature of the room. This room is the data centre and each data centre has hundreds of servers, hence the tremendous heat production and need for optimization of energy.

There are also two air flanges which represent the input and output flow of moist air flow through the room. They are represented by four properties which is pressure, mass fraction, enthalpy and mass flow rate of the moist air. With the help of mass and energy balances , which have been used to model the air compartment, the thermodynamic relation between input and output air and how the various heat exchanges inside the data centre effect these properties have been studied well, and are extremely useful for energy optimization.

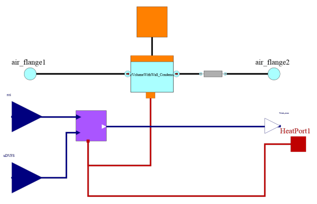


Figure 3.7: Block Diagram of Server With Air Compartment in Modelica

Simulations and Results

The results clearly indicate the increase of enthalphy 3.9 and absolute humidity 3.10 with the increase in the temperature of the server shown in Figure 3.8.

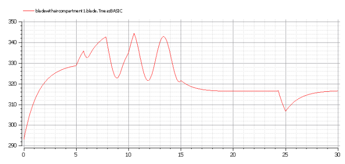


Figure 3.8: Temperature of a Server with Air Compartment

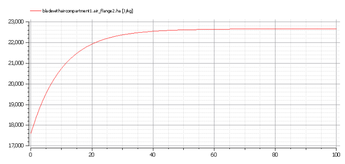


Figure 3.9: Specific Enthalpy increase due to the increasing temperature of the server

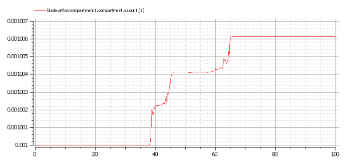


Figure 3.10: Increase in absolute humidity of dry air

3.3 Simulation and Implementation of the Data Centre

The working of the rack has been explained in the previous section. In this section, the modelica simulation and implementation of the rack is presented.

Block Diagram implementation in Modelica

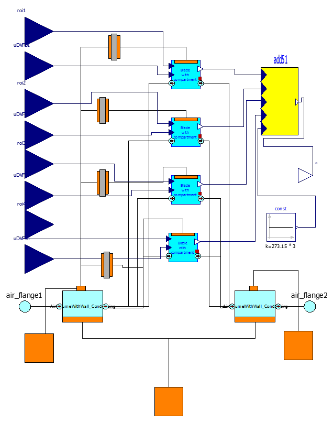


Figure 3.11: Servers arranged in a Rack Configuration

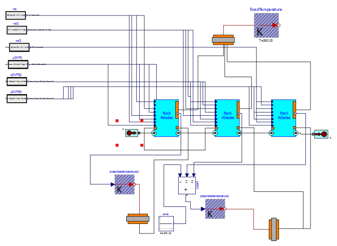


Figure 3.12: Racks arranged in a Data Centre Confguration

Simulation and Results

The results presented in Figure 3.13 and 3.14 show the behaviour of the data centre, which have been designed to have a cold aisle and a hot aisle. In Figure

3.13, the red graph shows the temperature of the cold aisle, while the green shows the temperature of the hot aisle

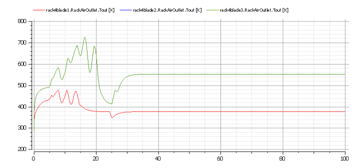


Figure 3.13: Output Temperature of Racks

Figure 3.14 shows the enthalpies of the three racks which form the data centre.

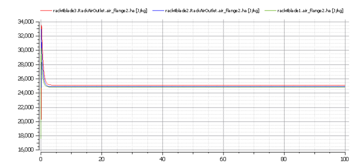


Figure 3.14: Enthaphy of Racks

3.4 Conclusion

This chapter successfully presented the implementation, simulation and results obtained when the models were simulated in modelica. The next chapter deals with the design of controllers for energy optimization of data centres by effective temperature control of servers.

Chapter 4

Controller Design

In the economy of the thesis, the role of this chapter is to

provide an overview of different controllers that can be used to govern the server thermal behaviour, and to use the presented server model for their setup, thereby assessing the model as suitable for the synthesis of such local controllers.

4.1 Introduction

Before presenting the design of the controller used in our project, it is important to shortly describe the history of why these controllers came into existence. Control theory is an interdisciplinary branch of engineering and mathematics dealing with the behaviour of dynamic systems with inputs. The objective of control theory is to calculate solutions for the proper corrective action from the controller that results in system stability, i.e., the system will hold the set point and not oscillate around it. It can be divided into two main branches, classical control and modern control.

Classical control theory is just limited to single input single output (SISO) while Modern control theory can deal with multi-input and multi-output (MIMO) systems. Hence, modern control theory overcomes the limitations of classical control theory in more sophisticated design problems. Two common types of control are the feedback control and the sequence control.

Feedback control is usually a continuous process and includes taking measurements with a sensor and making calculated adjustments via the controller to an output device to keep the measured variable within a set range. For instance, in a water heater, the sensor is the thermometer which measures the temperature of the water. The output of the thermometer is sent to the controller which compares the current temperature to the set point (aka desired temperature). Then, based on the difference between the current temperature and the set point a signal will be sent to the heaters to go on or off depending upon whether or not the water is hot enough or not. A simple Feedback controller is shown in Figure 4.1.

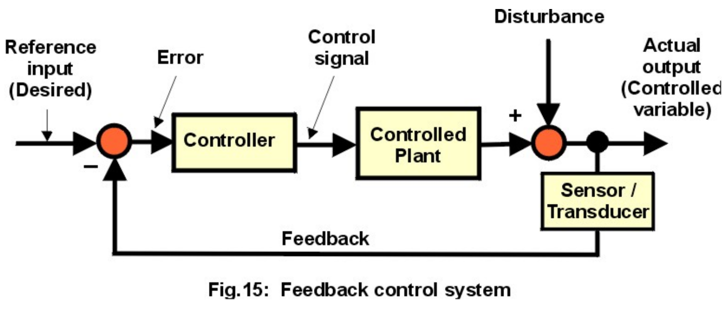

Reference input (Desired)
Error
Control signal
Disturbance
Actual output (Controlled variable)
+
−
Controller
Controlled Plant
Sensor / Transducer
Feedback
Fig.15: Feedback control system

Figure 4.1: A simple Feedback Controller

In a Feedback Controller, another important aspect to understand is whether it is open loop or closed loop .An Open-Loop controller does not have any measurement of the system's output – e.g., the water temperature – used to alter the water heating element. As a result, the controller cannot compensate for changes acting on the system. Open Loop controls are usually managed by human intervention where an operator observes a key metric– such as system power, pressure, or level – and then makes manual adjustments to the controls to achieve the desired result. In a Closed-Loop controller, as depicted in Figure 4.1 above, a sensor monitors the system's condition (e.g., temperature, pressure, speed, etc.) and feeds the data to a controller which adjusts the output device (e.g., the water heater heating element) as necessary to maintain the desired system output such as temperature, speed, etc. The design of this feedback process can also be referred to as a Control Loop since the system state is fed back to the controller and referenced to provide an error signal to the controller to make the necessary changes to the output device.

On the other hand, Sequence Control may be either to a fixed sequence or a logical one that performs different actions based on various system states.As sequential controls were established and became more and more part of the industrial automation landscape they became included in Relay Logic. Essentially this is where electrical relays engage electrical contacts which either start or interrupt power to a device. According to one source, electrical relays are referenced in industrial automation discussions as early as 1860. An example of a simple solar tracking sequence controller is shown in Figure 4.2

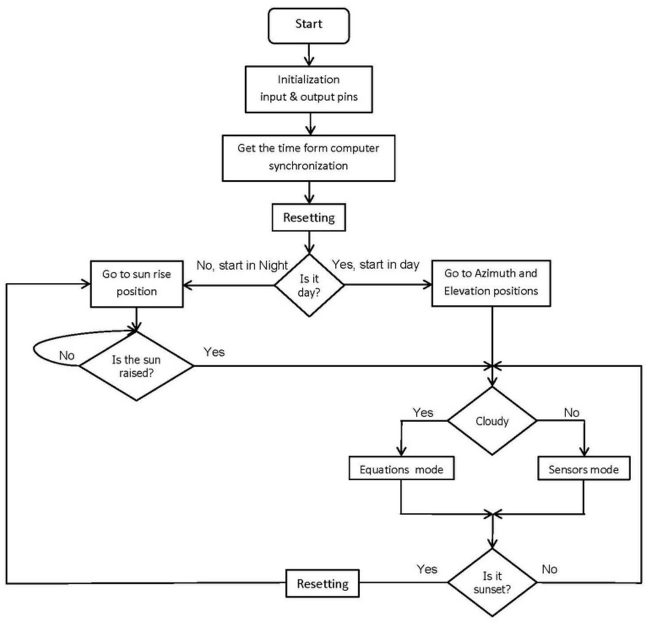


Figure 4.2: A simple Sequence Controller

4.2 Design of the Controller

The vast majority of all feedback controllers today are implemented using digital computers, relying on periodic sampling, computation, and actuation. For linear systems, sampled-data control theory provides powerful tools for direct digital design, while implementations of nonlinear control designs tend to rely on discretization combined with fast periodic sampling. In recent years, there has been a growing research interest in event-based control, in particular in connection to distributed and networked control systems.

The basic idea is to communicate, compute, or control only when something significant has occurred in the system. The motivation for abandoning the time-triggered paradigm is to better cope with various constraints or bottlenecks in the system, such as sensors with limited resolution, limited communication or computation bandwidth, energy constraints, or constraints on the number of actuations.

As compared to a fixed rate controller , where events are triggered periodically based on the sampling time , events based controller is a means to acquire measurements, take decisions and apply actions, "only when needed". The main differences between the way, these two kinds of controllers operate is based on the fact that in a fixed rate controller , the measurement of the control variable is taken at a constant rate, based on which the control signal is computed and that value is actuated, whereas an event based controller differs from the previous one in two ways, firstly the time between events is not constant and secondly , there can be multiple sources of events.

In this work thesis, firstly a continuous Proportional Integral (PI) controller has been used to control the temperature of the server by keeping the DVFS command under check. Then the same controller has been converted to a discrete form and used as a fixed rate controller. Finally , the discrete event controller has been implemented.

4.2.1 Continuous Proportional Integral Controller (PI) Design with Anti-Windup

PI controllers are one of the most widely used controllers. The combination of proportional and integral terms is important to increase the speed of the response and also to eliminate the steady state error. The anti-windup or saturation is important specially in cases where the control input needs to be kept bounded. The basic idea of this controller is that there is a reference value which needs to be tracked and the error is computed based on this reference value and the output value of the system to be controlled, which is fed back to the controller. The PI controller acts in such a way that this error becomes zero and the system successfully tracks the reference value. The combination of proportional and integral terms is important to increase the speed of the response and also to eliminate the steady state error. The proportional and integral terms are given by

$$u(t) = k_p * e(t) + k_i * \int e(t) * dt \tag{4.1}$$

where

e(t)=calculated error between the reference signal and the output.

k_p=Proportional gain of the PI controller

k_i=Integral gain of the PI controller

and

u(t)=controller signal which is input to the system

The block diagram of the PI controller used in our thesis is given in Figure 4.3.

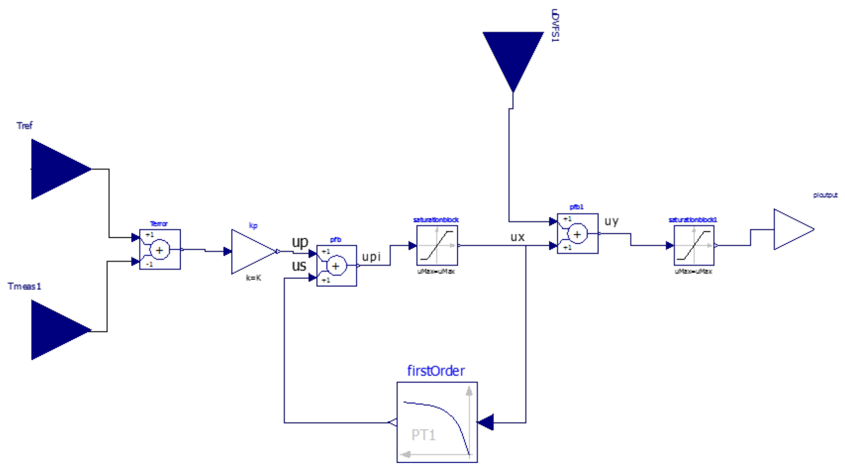


Figure 4.3: Block Diagram of PI Controller with anti-wind up

With reference to the block diagram presented in Figure 4.3, the equations representing the controller can be written as follows:

The error between the reference and the output from the system is given by

$$e_p = Trefpi - Tmeaspi \tag{4.2}$$

where

ep= error between the reference temperature and the output temperature of the system

and

Trefpi=reference temperature to be maintained by the controller

and

Tmeaspi= measured output of the system

The saturated control signal ux is given by

$$ux = us + Ti * \frac{d(us)}{dt} \tag{4.3}$$

$$us = \frac{ux}{1 + (s * Ti)} \tag{4.4}$$

where Ti= Integral time constant of the PI controller.

$$up = k_p * e_p \tag{4.5}$$

where k_p= *Proportional Gain of the PI controller.*

$$upi = up + us \tag{4.6}$$

where

upi= unsaturated control signal

$$ux = sat(upi) \tag{4.7}$$

$$uy = ux + DVFS \tag{4.8}$$

where

DVFS= Dynamic Voltage Frequency Scaling and is an input to the controller

$$ucontrolpi = sat(uy) \tag{4.9}$$

where

ucontrolpi= Final saturated ouput of the PI controller, which is the input of the system

Tuning Rule

For tuning a PI controller, certain criteria need to be kept under consideration.

- Integral Time Constant (Ti): Usually the integral time constant is set equal to the system time constant.

$$Ti = T \tag{4.10}$$

where

T=time constant of the system to be controlled

- Proportional Gain *(k_p)*: The proportional gain of the PI controller is given by

$$k_p = \frac{w_p * Ti}{G} \tag{4.11}$$

where

w_p=Cut off frequency of the system chosen according to requirements

and

G=Gain of the system to be controlled

Coding and Block Diagram

The modelica coding for the design of a continous PI controller is given in Code Snippet 4.1.

```
model pimodelcoontinous

  parameter Real Kp = 1000 "Kp";
  parameter SI.Time Ti = 0.1 "Ti";
  parameter Real uMax(start = 1) = 1 "Upper limits of input signals";
  parameter Real uMin = -1 "Lower limits of input signals";
  parameter Boolean strict = false "= true, if strict limits with noEvent(..)" annotation(Evaluate = true, choices(checkBox = true), Dialog(tab = "Advanced"));
  parameter Boolean limitsAtInit = true "= false, if limits are ignored during initialization (i.e., y=u)" annotation(Evaluate = true, choices(checkBox = true), Dialog(tab = "Advanced"));

  Real ep;
  Real up;
  Real us;
  Real ux;
  Real uy;
  Real upi;

  Modelica.Blocks.Interfaces.RealInput uDVFSPI annotation(Placement(visible = true, transformation(origin = {-80, 20}, extent = {{-10, -10}, {10, 10}}, rotation = 0), iconTransformation(origin = {-17.5, 82.5}, extent = {{-17.5, -17.5}, {17.5, 17.5}}, rotation = -90)));
  Modelica.Blocks.Interfaces.RealInput Trefpi annotation(Placement(visible = true, transformation(origin = {-95, 55}, extent = {{-15, -15}, {15, 15}}, rotation = 0), iconTransformation(origin = {-82.5, 37.5}, extent = {{-17.5, -17.5}, {17.5, 17.5}}, rotation = 0)));
  Modelica.Blocks.Interfaces.RealInput Tmeaspi annotation(Placement(visible = true, transformation(origin = {-95, -25}, extent = {{-15, -15}, {15, 15}}, rotation = 0), iconTransformation(origin = {-80, -50}, extent = {{-20, -20}, {20, 20}}, rotation = 0)));
  Modelica.Blocks.Interfaces.RealOutput ucontrolpi annotation(Placement(visible = true, transformation(origin = {80, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0), iconTransformation(origin = {85, -5}, extent = {{-15, -15}, {15, 15}}, rotation = 0)));
equation
  ep = Trefpi - Tmeaspi;
  //equation 1
  up = ep * Kp;
  //equation 2
  us = ux - Ti * der(us);
  // equation 3
  upi = up + us;
  // equation 4
  assert(uMax >= uMin, "Limiter: Limits must be consistent. However, uMax (=" + String(uMax) + ") < uMin (=" + String(uMin) + ")");
  if initial() and not limitsAtInit then
```

```
    ux = upi;
    // equation 5
    assert(upi >= uMin - 0.01 * abs(uMin) and upi <= uMax + 0.01 * abs(uMax), "
    Limiter: During initialization the limits have been ignored.\n" + "However,
    the result is that the input upi is not within the required limits:\n" + "
    upi = " + String(upi) + ", uMin = " + String(uMin) + ", uMax = " + String(
    uMax));
  elseif strict then
    ux = smooth(0, noEvent(if upi > uMax then uMax else if upi < uMin then uMin
    else upi));
  else
    ux = smooth(0, if upi > uMax then uMax else if upi < uMin then uMin else upi)
    ;
  end if;
  if ep <= 0 then
    uy = ux + uDVFSPI;
  else
    uy = uDVFSPI;
  end if;
  //equation 6
  assert(uMax >= uMin, "Limiter: Limits must be consistent. However, uMax (=" +
    String(uMax) + ") < uMin (=" + String(uMin) + ")");
  if initial() and not limitsAtInit then
    ucontrolpi = uy;
    // equation 7
    assert(uy >= uMin - 0.01 * abs(uMin) and uy <= uMax + 0.01 * abs(uMax), "
    Limiter: During initialization the limits have been ignored.\n" + "However,
    the result is that the input uy is not within the required limits:\n" + " uy
     = " + String(uy) + ", uMin = " + String(uMin) + ", uMax = " + String(uMax));
  elseif strict then
    ucontrolpi = smooth(0, noEvent(if uy > uMax then uMax else if uy < uMin then
    uMin else uy));
  else
    ucontrolpi = smooth(0, if uy > uMax then uMax else if uy < uMin then uMin
    else uy);
  end if;

end pimodelcoontinous;
```

Code Snippet 4.1: Modelica Code Implementation of a Continous PI Controller

The block diagram implementation of the system along with the controller in given in Figure 4.4. A short explanation of the way the controller has been designed and how it is useful to make the system behave in the desired way is deemed necessary in the scope of the work carried out.

Firstly, the system is the server, which has two inputs DVFS and roi and one output which is the temperature of the server. The main aim of the controller is to not let the output temperature of the server exceed the operating limits, which can be detrimental to the working of the server.

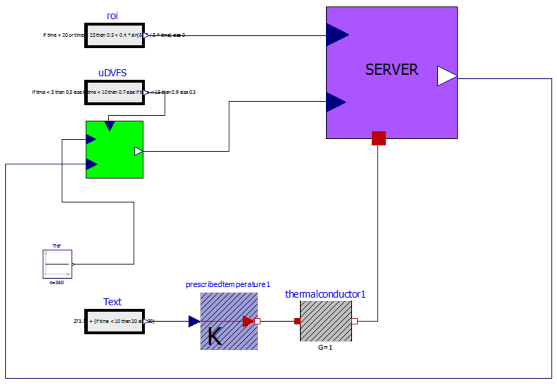


Figure 4.4: block diagram implementation of the continuous PI Controller plus the system

In case, the temperature of the server exceeds the reference range, the controller acts in such a way that it will make the system follow the reference value by acting on the DVFS command and if the temperature of the system is below the reference, then the controller doesn't take any action and let the system operate in open loop. In this case, the DVFS is considered to be the input of the system and output of the controller while roi is considered as a disturbance.

Simulation and Results

The results of the simulations are shown in the following figures. Figure 4.5 and 4.6 show the open loop response and the controlled response of the server respectively.In this case, the reference value has been set to 360 Kelvin. As can be observed from the results,that as long as the open loop temperature is below the reference, the controller does not come into action but once the temperature of the system goes above the reference, the controller makes sure that it starts tracking the reference.

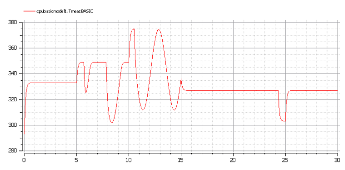


Figure 4.5: Open loop temperature response of the Server (Kelvin)

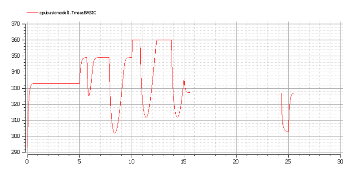


Figure 4.6: Controlled Temperature of the Server Tracking the reference at 360 Kelvin (Kelvin)

Figure 4.7 and 4.8 shows the DVFS command in case of open loop server and when a controller is included in the loop respectively.

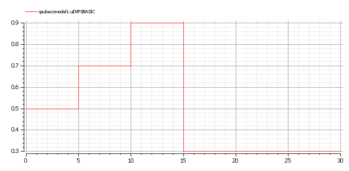


Figure 4.7: Open loop DVFS input of the Server

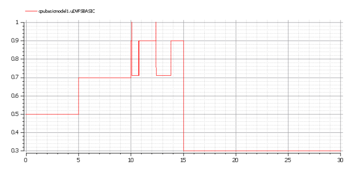


Figure 4.8: Controlled DVFS input of the system which is the output of the controller

Figure 4.9 and 4.10 show the open loop power and the power in case of the controller which is consumed respectively.

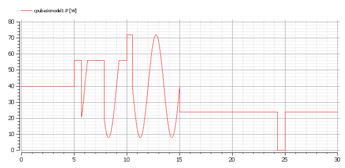


Figure 4.9: Open loop power response of the Server (Watts)

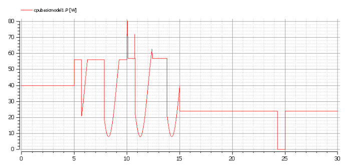


Figure 4.10: Power Consumed in case of the temperature controlled server (Watts)

4.2.2 Fixed Rate Discrete Proportional Integral Controller (PI) Design with Anti-Windup

Since the final aim is to design a discrete event based controller, first step is to discretize the controller and apply it to fixed rate controller to test if it is working properly. It is important to note that fixed-rate controller necessarily yields the same results as continuous controller which will be evidenced while demonstrating results.

First step in the design of a fixed rate controller is discretization.

Discretization

There are many different types of discretization methods used. The simplest way to convert any system G(s) defined in s-domain to G(z)in z-domain is to find transform which relates one to another. Consider the problem of integrating a generic signal, x(t). In continuous time, if x is integrated , it has the laplace transform of $\frac{1}{s}$. Defining this equation by y(t), it can be written as

$$y(t) = \int x * dt \tag{4.12}$$

In software, it can be approximated as

$$y = y + x * dt \tag{4.13}$$

With Euler Integration, the relationship between s and z is then

$$y(k) = y(k-1) + u(k) * T \tag{4.14}$$

where

T=sampling time

Converting to z-domain

$$Y = \frac{1}{z} * Y + T * U \tag{4.15}$$

$$(z-1) * Y = T * z * U \tag{4.16}$$

$$Y = \frac{T * z}{z-1} * U \tag{4.17}$$

$$\frac{1}{s} = \frac{T * z}{z-1} \tag{4.18}$$

which results in

$$s = \frac{z-1}{T * z} \tag{4.19}$$

This equation 4.19 represents the Euler Integration Approximation

With reference to the block diagram shown in Figure 4.3, the Euler Integration Approximation obtained in equation 4.19 is applied to the continuous controller to convert it into discrete form.
Recalling equation 4.4 and substituting equation 4.19 into it,

$$us(z) = \frac{1}{1 + (\frac{z-1}{Ts*z}) * Ti} * ux(z) \tag{4.20}$$

$$\frac{us(z)}{ux(z)} = \frac{z * \frac{Ts}{Ts+Ti}}{z - \frac{Ti}{Ti+Ts}} \tag{4.21}$$

For simplification

$$a = \frac{Ti}{Ti + Ts} \tag{4.22}$$

and

$$b = \frac{Ts}{Ts + Ti} \tag{4.23}$$

which results in equation 4.21 becoming

$$\frac{us(z)}{ux(z)} = \frac{z * b}{z - a} \tag{4.24}$$

Using the partial fraction method , since we are dealing with not strictly proper systems

$$us(z) = b * ux(z) + \frac{a * b}{z - a} * ux(z) \tag{4.25}$$

and
representing the fraction from equation 4.25 by

$$s(k) = \frac{a * b}{z - a} * ux(k) \tag{4.26}$$

which results in equation 4.25 becoming

$$us(k) = s(k) + b * ux(k) \tag{4.27}$$

where

$$s(k) = a * s(k - 1) + a * b * ux(k - 1) \tag{4.28}$$

This is the discretized form of our controller, whose implementation has been shown in the next sections.

Coding and Block Diagram

This section shows the coding and block implementation in modelica to achieve the desired results.

```
model pidiscretefixedstep

  parameter Real Kp = 1000 "Kp";
  parameter SI.Time Ti = 0.1 "Ti";
  parameter Real uMax = 1 "Upper limits of input signals";
  parameter Real uMin = -1 "Lower limits of input signals";
  parameter Real ustart = 0;
  parameter Real Ts = 0.01;
  discrete Real TrefpilastComp;
  discrete Real TmeaspilastComp;
  discrete Real uDVFSPIlastComp;
  // discrete Real tLastComp;
  discrete Real u;
  discrete Real us;
  discrete Real ux;
  discrete Real uxold;
  discrete Real skold;
  discrete Real uy;
  //discrete Real ud;
  discrete Real sk;
  discrete Real a;
  discrete Real b;
  discrete Real ep;
  discrete Real upi;
  //discrete Real tLastComp;

  Modelica.Blocks.Interfaces.RealInput Trefpidfix annotation(Placement(visible =
    true, transformation(origin = {-100, 60}, extent = {{-10, -10}, {10, 10}},
    rotation = 0), iconTransformation(origin = {-85, 55}, extent = {{-15, -15}, {
    15, 15}}, rotation = 0)));
  Modelica.Blocks.Interfaces.RealInput Tmeaspidfix annotation(Placement(visible =
     true, transformation(origin = {-100, -20}, extent = {{-10, -10}, {10, 10}},
    rotation = 0), iconTransformation(origin = {-85, -25}, extent = {{-15, -15},
    {15, 15}}, rotation = 0)));
  Modelica.Blocks.Interfaces.RealInput uDVFSPIdfix annotation(Placement(visible =
     true, transformation(origin = {0, 100}, extent = {{-10, -10}, {10, 10}},
    rotation = -90), iconTransformation(origin = {-5, 85}, extent = {{-15, -15},
    {15, 15}}, rotation = -90)));
  Modelica.Blocks.Interfaces.RealOutput ucontrolpid annotation(Placement(visible
    = true, transformation(origin = {100, 20}, extent = {{-10, -10}, {10, 10}},
    rotation = 0), iconTransformation(origin = {85, 15}, extent = {{-15, -15}, {
    15, 15}}, rotation = 0)));
equation
  ucontrolpid = u;
  //uDVFSPI = ud;
algorithm
  when initial() then
    TrefpilastComp := Trefpidfix;
    TmeaspilastComp := Tmeaspidfix;
    uDVFSPIlastComp := uDVFSPIdfix;
    ux := ustart;
    uxold := ux;
    skold := sk;
    uxold := 0;
    skold := 0;
    us := ux;
  end when;
  //us:=ux;
  //when sample(0, Ts) then
  when sample(0, Ts) and not initial() then
    ep := Trefpidfix - Tmeaspidfix;
    upi := Kp * ep + us;
    b := Ts / (Ts + Ti);
    a := Ti / (Ts + Ti);
    us := b * ux + sk;
    sk := a * skold + a * b * uxold;
    ux := max(uMin, min(uMax, upi));
    if ep <= 0 then
      uy := uDVFSPIdfix + ux;
    else
      uy := uDVFSPIdfix;
    end if;
    u := max(uMin, min(uMax, uy));
    TrefpilastComp := Trefpidfix;
    TmeaspilastComp := Tmeaspidfix;
    uDVFSPIlastComp := uDVFSPIdfix;
    uxold := ux;
    skold := sk;
  end when;
  //when not initial() then

end pidiscretefixedstep;
```

Code Snippet 4.2: Modelica Code Implementation of a Fixed Rate Discrete PI Controller

The block diagram shown in Figure 4.11 shows the implementation of fixed rate controller to control the output temperature of the server.

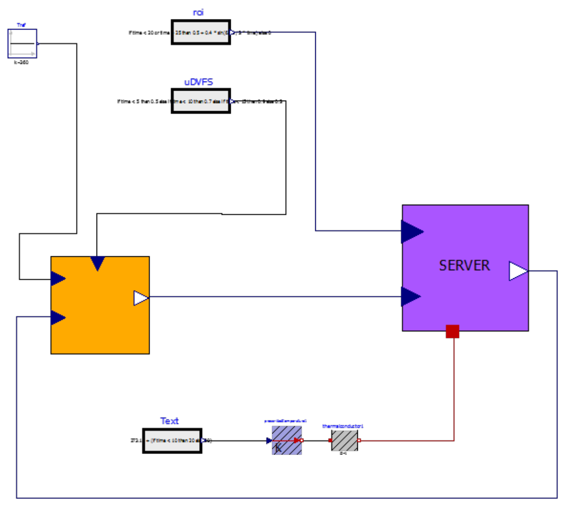


Figure 4.11: Modelica Block Diagram of a discrete PI controller

Simulations and Results

As can be observed from figure 4.12, the fixed rate controller successfully controls the temperature of the server and yields the same result as that of the continuous proportional controller with reference to figure 4.6.

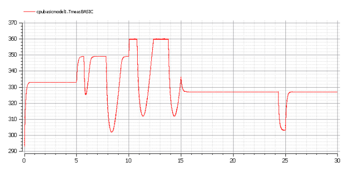


Figure 4.12: Modelica Block Diagram of a fixed step discrete PI controller

Figure 4.13 shows the sampling time of the fixed step discrete PI controller and as can be seen, it is fixed for this kind of controller.

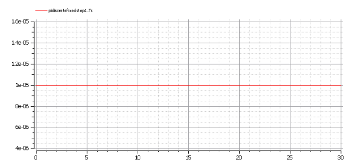


Figure 4.13: Sampling time of a fixed step discrete PI controller

4.2.3 Discrete Event Based Controller

This section will provide a short description of the design of discrete events controller, which is important in the scope of the project. In this work, some industrially realistic hypotheses that lead to consider a class of event-based control systems that is in some sense the closest to the fixed-rate case is introduced. Firstly, the hypothesis which is the basis of design of the event based controller is designed.

General Hypothesis

For simplicity , the case in which event based information flow originates from the sensor is considered. The hypothesis are as follows:

- Hypothesis 1: The system to be controlled is linear time invariant (LTI)single input, single output(SISO)system. The model has to be strictly proper.
- Hypothesis 2: A continuous LTI SISO controller that stabilizes the nominal closed loop system containing model is available. In case of this project, PI controller has been chosen.
- Hypothesis 3: The controller mentioned in Hypothesis 2 is realized with digital technology and computes the discrete-time control u*(k) at events, which occur at time instants t_k counted by an integer k € N, and not evenly spaced.
- Hypothesis 4: Events are triggered by a single source (assumed to be sensor).
- Hypothesis 5: The time between two events is an integer multiple of a quantum q_s € R, $q_s > 0$.

$$\forall\ t_h \leq t_k, t_k - t_h = \zeta(k,h) q_s,\ \zeta_{k,h} \text{ € } N.$$

According to Hypothesis 4, two quantities can be defined

  - the a priori step duration $\overline{T}_s$(k) that is decided at the k-th event,
  - the a posteriori step duration $\underline{T}_s$(k), i.e., the time actually elapsed from the k-th to the (k + 1)-th event.

  In practice, events can occur at the termination of $\overline{T}_s$(k) or earlier, no matter why. In the former case $\underline{T}_s$(k)=$\overline{T}_s$, while in the latter $\underline{T}_s$ (k) < $\overline{T}_s$(k). Notice that the event generation mechanism is in part reactive and in part proactive, the timeout being in fact the simplest way to decide when the next event has to occur.

- Hypothesis 6:There is an upper bound for the time between two subsequent events, i.e.,

$$\forall \text{ k}, \sigma(k) € \sum = [1, ..., N] \subset \text{N}, 1\leq\text{N}< \infty$$
$$\text{where}$$
$$\sigma(k):=\zeta(k+1,k)$$

- Hypothesis 7: The control signal is kept constant between two subsequent events, as in the extremely frequent case where a zero order holder is used.

- Hypothesis 8: When an event is triggered by the sensor, this results in the computation and actuation of a new control value. The delay between the triggered event and the control actuation is either negligible or known and constant, so that it can be taken as a part of the process model.

Triggering Rule

For the application of the triggering rule, it is very important to have a stable system. Once a stable system has been ensured, the triggering rule can be applied. Basically, one wants the (a posteriori) control step duration

- to increase as rapidly as possible toward its allowed maximum if the sensor triggers no event, which typically occurs with the "send on delta" policy, i.e., when the controlled variable, polled by the sensor at rate 1/q$_s$, differs in magnitude from the last transmitted one by more than a prescribed amountδ y.

- to allow reacting as soon as possible to an event, the minimum reaction time being q$_s$.

- and to avoid event hauls after the first one triggered by a controlled variable's variation.

To achieve that, once q_s and N are decided, a subset $\overline{\sum} = [\overline{\sigma_i}]$ of the $\sum$ is defined, with cardinality $\overline{N}$ <N, so that $\overline{\sigma_1}$ be greater than 1, $\overline{\sigma_1}$ q_s be a "small but reasonable" sampling period if adopted for a fixed-rate controller realization, and $\overline{\sigma_N}$ = N.
The a priori period $\overline{T}_s$ is first initialised to $\overline{\sigma_1}$ q_s. Then, if a step of a priori duration $\overline{\sigma_i}$ q_s elapses, the next a priori period is set to $\overline{\sigma_{i+1}}$ q_s, until $\overline{\sigma_N}$ is reached. If conversely a step ends due to a sensor event, the next period is reset to $\overline{\sigma_1}$ q_s and the system is forced to make it elapse. This temporary constraint results in possibly ignoring some events, which is however harmless because it was just stated that if the controller were realised as a fixed-rate one with sampling period $\overline{\sigma_1}$ q_s, the consequent latency – e.g., in reacting to a disturbance – would be acceptable.

Coding and Block Diagram

Modelica code implementation of Discrete Events based controller is shown in Code Snippet 4.3. As can observed from the code, as compared to fixed rate controller, where the sampling time is fixed, in this case the the sampling time depends on the events generation.

```
model picontroldiscrete

  parameter Real Kp = 1000 "Kp";
  parameter SI.Time Ti = 0.1 "Ti";
  parameter Real uMax = 1 "Upper limits of input signals";
  parameter Real uMin = -1 "Lower limits of input signals";
  parameter Real ustart = 0;
  Real Ts;
  discrete Real TrefpilastComp;
  discrete Real TmeaspilastComp;
  discrete Real uDVFSPIlastComp;
  discrete Real tLastComp;
  discrete Real u;
  discrete Real us;
  discrete Real ux;
  discrete Real uxold;
  discrete Real uy;
  //discrete Real ud;
  discrete Real sk;
  discrete Real skold;
  discrete Real a;
  discrete Real b;
  discrete Real ep;
  discrete Real upi;

  Modelica.Blocks.Interfaces.BooleanInput Etrig annotation(Placement(visible =
    true, transformation(origin = {-100, 20}, extent = {{-10, -10}, {10, 10}},
    rotation = 0), iconTransformation(origin = {-85, -5}, extent = {{-15, -15}, {
    15, 15}}, rotation = 0)));
  Modelica.Blocks.Interfaces.RealInput Tmeaspid annotation(Placement(visible =
    true, transformation(origin = {-100, -40}, extent = {{-10, -10}, {10, 10}},
    rotation = 0), iconTransformation(origin = {-85, -65}, extent = {{-15, -15},
    {15, 15}}, rotation = 0)));
  Modelica.Blocks.Interfaces.RealInput uDVFSPId annotation(Placement(visible =
    true, transformation(origin = {-20, 100}, extent = {{-10, -10}, {10, 10}},
    rotation = -90), iconTransformation(origin = {-15, 85}, extent = {{-15, -15},
     {15, 15}}, rotation = -90)));
  Modelica.Blocks.Interfaces.RealInput Trefpid annotation(Placement(visible =
    true, transformation(origin = {-100, 60}, extent = {{-10, -10}, {10, 10}},
    rotation = 0), iconTransformation(origin = {-85, 55}, extent = {{-15, -15}, {
    15, 15}}, rotation = 0)));
  Modelica.Blocks.Interfaces.RealOutput ucontrolpid annotation(Placement(visible
    = true, transformation(origin = {100, 20}, extent = {{-10, -10}, {10, 10}},
    rotation = 0), iconTransformation(origin = {85, 15}, extent = {{-15, -15}, {
    15, 15}}, rotation = 0)));

equation
  ucontrolpid = u;
  //uDVFSPI = ud;
algorithm
  when initial() then
    TrefpilastComp := Trefpid;
    TmeaspilastComp := Tmeaspid;
    uDVFSPIlastComp := uDVFSPId;
    ux := ustart;
    uxold := ux;
    skold := sk;
    uxold := 0;
    skold := 0;
    us := ux;
  end when;
  when Etrig <> pre(Etrig) and not initial() then
    Ts := time - tLastComp;
    tLastComp := time;
    ep := Trefpid - Tmeaspid;
    upi := Kp * ep + us;
    ux := max(uMin, min(uMax, upi));
    b := Ts / (Ts + Ti);
    a := Ti / (Ts + Ti);
    us := b * ux + sk;
    sk := a * skold + a * b * uxold;
    if ep <= 0 then
      uy := uDVFSPId + ux;
    else
      uy := uDVFSPId;
    end if;
    u := max(uMin, min(uMax, uy));
    TrefpilastComp := Trefpid;
    TmeaspilastComp := Tmeaspid;
    uDVFSPIlastComp := uDVFSPId;
    uxold := ux;
    skold := sk;
  end when;
  //us:=(Ti * pre(us) + Ts * ux) / (Ti + Ts);
  //uy:=pre(uy) + uDVFSPI + ux + pre(ux);
```

Code Snippet 4.3: Modelica Code Implementation of a Discrete Events Based PI Controller

In code snippet 4.4 , the coding and implementation of trigger for generation events is shown.

```
model trigger
  Modelica.Blocks.Interfaces.RealInput Tmeastrig annotation(Placement(visible =
    true, transformation(origin = {-95, 15}, extent = {{-15, -15}, {15, 15}},
    rotation = 0), iconTransformation(origin = {-100, 20}, extent = {{-10, -10},
    {10, 10}}, rotation = 0)));
  parameter Real DeltaTmeastrig = 0.01 "Threshold for send on Delta";
  parameter Real qs = 0.1 "Ts quantum (internal sensor sampling time)";
  parameter Real qsm[:] = {10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000} "
    qs multipliers (last elem is N)";
  parameter Real pTmeastrigfilt = 0.5 "pole of the DT PV filter";
  //paramter Real Tref = 360 "Tref";
  discrete Boolean Egend;
  discrete Boolean wait4Ts;
  discrete Real Tmeastrigd;
  discrete Real TmeastriglastCom;
  discrete Real tLastCom;
  discrete Real Ts;
  discrete Integer EvCnt;
  discrete Integer dex;
  Modelica.Blocks.Interfaces.BooleanOutput Egen annotation(Placement(visible =
    true, transformation(origin = {95, 15}, extent = {{-15, -15}, {15, 15}},
    rotation = 0), iconTransformation(origin = {100, 20}, extent = {{-10, -10}, {
    10, 10}}, rotation = 0)));
equation
  Egen = Egend;
algorithm
  when initial() then
    TmeastriglastCom := Tmeastrig;
    tLastCom := 0;
    Egend := false;
    wait4Ts := false;
    dex := 1;
    Ts := qsm[1] * qs;
    EvCnt := 0;
  end when;
  when sample(0, qs) then
    Tmeastrigd := pTmeastrigfilt * pre(Tmeastrigd) + (1 - pTmeastrigfilt) *
    Tmeastrig;
    if abs(Tmeastrigd - TmeastriglastCom) >= DeltaTmeastrig and not wait4Ts or
    time - tLastCom >= Ts then
      if time - tLastCom >= Ts then
        dex := min(dex + 1, size(qsm, 1));
        wait4Ts := false;
      else
        dex := 1;
        wait4Ts := true;
      end if;
      Ts := qsm[dex] * qs;
      TmeastriglastCom := Tmeastrig;
      tLastCom := time;
      Egend := not Egend;
      if time >= 0 then
        EvCnt := EvCnt + 1;
      end if;
    end if;
  end when;

initial algorithm
  Tmeastrigd := Tmeastrig;
end trigger;
```

Code Snippet 4.4: Modelica Code Implementation of the Trigger used in Discrete Events Based PI Controller

The block diagram of discrete events based controller is shown in Figure 4.14. The yellow block is the trigger and is connected to the controller through a boolean signal. Based on that signal, events are generated and controller works to make the system work in the desired way.

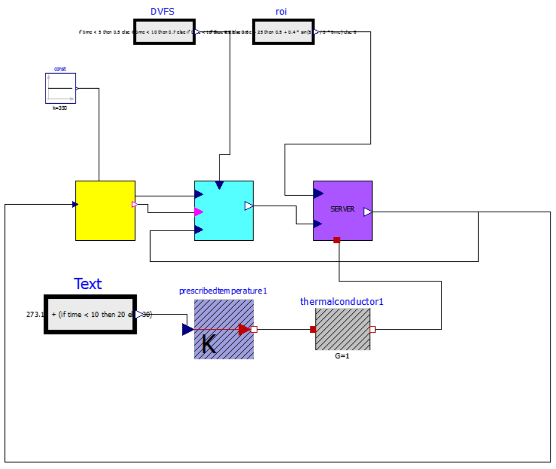


Figure 4.14: Modelica Block Diagram of a Discrete Event based PI controller

Simulation and Results

This section shows the results which have been obtained by successful implementation of the discrete event based controller to the system which is the server. The reference temperature to be tracked is kept at 350 kelvin and the controlled system successfully tracks the reference with the controller only acting at times when the temperature exceeds the reference temperature.

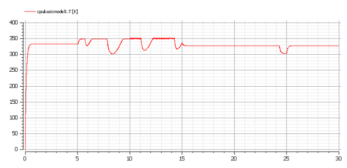


Figure 4.15: Controlled temperature of the server using discrete events based controller

As can be seen in Figure 4.16, the sampling time in case of an events based controller is no longer constant.

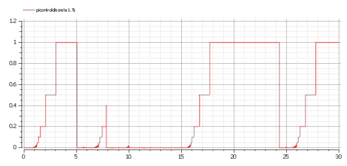


Figure 4.16: Sampling time of a discrete event based controller

4.2.4 Linear Quadratic Regulator (LQR)

Linear Quadratic Regulator is a special controller and belongs to the family of optimal control. Optimal control deals with the problem of finding a control law for a given system such that a certain optimality criterion is achieved. A control problem includes a cost function that is a function of state and control variables. An optimal control is a set of differential equations describing the paths of the control variables that minimize the cost functional. The optimal control can be derived using Pontryagin's maximum principle (a necessary condition also known as Pontryagin's minimum principle or simply Pontryagin's Principle), or by solving the Hamilton–Jacobi–Bellman equation (a sufficient condition).The theory of optimal control is concerned with operating a dynamic system at minimum cost. The case where the system dynamics are described by a set of linear differential equations and the cost is described by a quadratic function is called the LQ problem. One of the main results in the theory is that the solution is provided by

the linear-quadratic regulator (LQR), a feedback controller. The LQR is one of the most fundamental problems in control theory.
Based on the different ways to define a cost function, LQR problem can be of different types most commonly finite horizon continuous time, infinite horizon continuous time, finite horizon discrete time and infinite horizon discrete time LQR. In this case , infinite horizon continuous time LQR has been used.The equations defining the LQR controller are:

The general linear system to be controlled is given in the form of

$$\dot{x} = [A]x + [B]u \tag{4.29}$$

where
x=states of the system
u=inputs of the system
*[A] =open loop eigenvalues of the system and is a n*1 matrix, in case of n=1 is a scalar.*
*[B]=p*1 matrix, in case of p=1 is a scalar*
For a continuous time system, the state feedback law u=r-kx minimizes the quadratic cost function

$$J(u) = \int_0^\infty (x^T * Q * x + u^T * R * u)dt \tag{4.30}$$

subject to the system dynamics in equation 4.29
where
J=Quadratic cost function
r=reference signal to be tracked by the controller
k=controller gain to stabilize the system.
*[Q]= is a positive definite matrix and is the gain to penalize states of the system and is a n*n matix*
and
*[R]= is a positive definite matrix and is the gain to penalize the inputs of the system and is a p*p matrix*

Usually in order to find the desired eigenvalues, can take r=0, so that the it becomes the problem of a pure regulator . It turns out that the system eigenvalues that make the regulator work well are the same system values that make a tracker work when a reference other than zero is given. So can think of the problem as just a regulator problem with no reference. In a normal state feedback controller, when placing the close loop eigenvalues is that it is hard to have a great sense of dynamic response associated with the eigenvalues and then it is not always possible for the actuator to provide such high control inputs to give the desired eigenvalues. So how and where to place the eigenvalues is not very intuitive and that is what the LQR is about.

The feedback control law that minimizes the value of the cost is given by

$$u = -kx \tag{4.31}$$

where

$$k = R^{-1} * B^T * P \tag{4.32}$$

and

P is given by the algebraic riccati continuous equation (ARE) associated with the cost function

$$A^T * P + P * A + Q - P * B * R^{-1} * B^T = 0 \tag{4.33}$$

So the controlled tracking system is given by

$$\dot{x} = [A - Bk]x + [B]r \tag{4.34}$$

Coding and Block Diagram

Code Snippet 4.5 shows the modelica code used to model the LQR.

```
model LQRcontroller
  parameter Real A = -10;
  parameter Real B = 800;
  parameter Real Q = 1;
  parameter Real R = 0.25;
  Real k;
  Real P;
  Real ep;
  Modelica.Blocks.Interfaces.RealInput uDVFS annotation(Placement(visible = true,
     transformation(origin = {0, 100}, extent = {{-22, -22}, {22, 22}}, rotation
    = -90), iconTransformation(origin = {4, 78}, extent = {{-22, -22}, {22, 22}},
     rotation = -90)));
  Modelica.Blocks.Interfaces.RealInput Tlqr annotation(Placement(visible = true,
    transformation(origin = {-100, -50}, extent = {{-22, -22}, {22, 22}},
    rotation = 0), iconTransformation(origin = {-79, -63}, extent = {{-21, -21},
    {21, 21}}, rotation = 0)));
  Modelica.Blocks.Interfaces.RealOutput ulqr annotation(Placement(visible = true,
     transformation(origin = {99, 11}, extent = {{-19, -19}, {19, 19}}, rotation
    = 0), iconTransformation(origin = {81, 1}, extent = {{-19, -19}, {19, 19}},
    rotation = 0)));
  Modelica.Blocks.Interfaces.RealInput Treflqr annotation(Placement(visible =
    true, transformation(origin = {-94, 70}, extent = {{-10, -10}, {10, 10}},
    rotation = 0), iconTransformation(origin = {-80, 60}, extent = {{-20, -20}, {
    20, 20}}, rotation = 0)));
equation
  A * P + P * A + Q - P * B * (1 / R) * B * P = 0;
  k = B * (1 / R) * P;
  ep = Treflqr - Tlqr;
  if ep <= 0 then
    ulqr = Treflqr - k * Tlqr + uDVFS;
  else
    ulqr = uDVFS;
  end if;
  //ulqr = uDVFS;
  //ulqr = Treflqr - k * Tlqr + uDVFS;
end LQRcontroller;
```

Code Snippet 4.5: Modelica Code Implementation of the Linear Quadratic controller (LQR)

Figure 4.17 present the block diagram implementation of LQR to control the temperature of the server. There are two inputs roi and DVFS, the LQR controller has DVFS as its output while roi is considered as a disturbance.

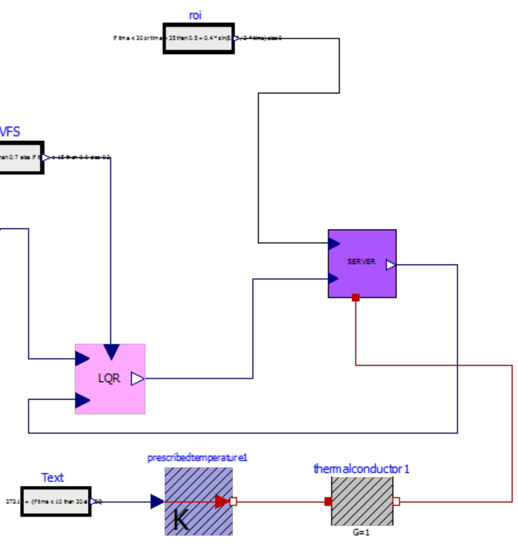


Figure 4.17: Block diagram of LQR implementation

Simulations and Results

This section presents the results achieved upon successful implementation of LQR to control the temperature of the server. Figure 4.3 and 4.19 show the open loop and the closed loop temperature response of the server. As can be seen from Figure 4.19, as long as the temperature is below the reference temperature, the control lets the system operate in open loop but as soon as the temperature crosses the reference, the controller starts operating and makes the system to track the reference temperature which has been set to 360 Kelvin in this case.

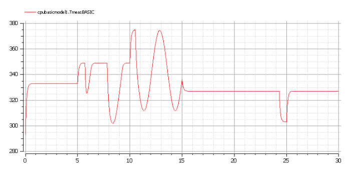


Figure 4.18: Open loop temperature response of the Server (Kelvin)

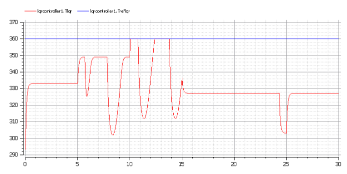


Figure 4.19: Controlled Temperature of the Server Tracking the reference at 360 Kelvin

Figure 4.20 and 4.21 show the open loop input to the system and the controlled input to the system respectively. As can be seen from Figure 4.21, the LQR gives the optimal input to the system.

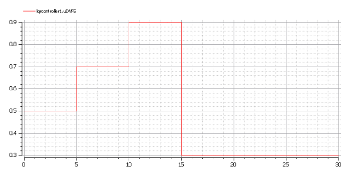


Figure 4.20: Open loop DVFS input of the server

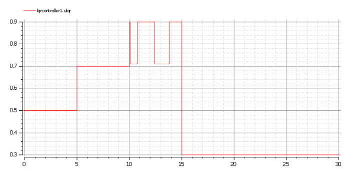


Figure 4.21: Controlled DVFS input of the system which is the output of the controller

Finally, Figure 4.22 and 4.23 compares the power consumption of the server in open loop and in case of LQR controlled server which reduces the power consumption consumption considerably.

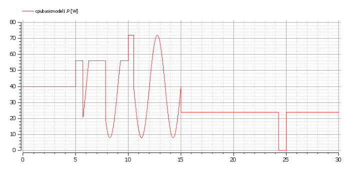


Figure 4.22: Open loop power response of the Server (Watts)

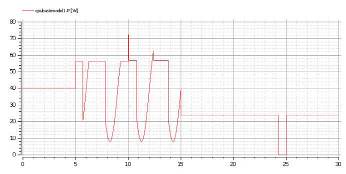


Figure 4.23: Power consumed in case of the temperature controller server(Watts)

Comparison of results of LQR with the PI controller presented in subsection 4.2.1

This section provides a comparison of the results obtained in case of PI controller and LQR with reference to the results presented in subsections 4.2.1 and 4.2.4. With reference to Figure 4.8 and 4.21, it can be easily noticed that LQR provides optimal input (DVFS) to the system to track the same reference as compared to PI controller and hence implies that to track the same output LQR needs less actuator efforts as compared to PI controller which in itself proves its optimality.

4.3 Conclusion

This chapter has given an overview of the different controllers which have been used to control the temperature of the server, their methodologies and the results obtained. The temperature control of the server was successfully achieved, and the presented model proved suitable for the setup of such control solutions.

Chapter 5

Conclusion and Future Works

This main objectives of the project have been successfully achieved. As can be observed from the state of art presented in chapter 1, this method of modelling and control has never been used before.Most of the methods presented till date have been focusing mostly on different ways to scale voltage and frequency of the processor of the servers by using different approaches for DVFS to optimize the computing power of these servers to reduce the energy consumption in these Data centres. The modelling of data centres from a system and control point of view using the basic principles of thermodynamics and simple power equations is a completely new approach as compared to how the computer science deals with the problem and has the potential of producing far better results in terms of power optimization. This method is also useful in preventing unnecessary loops for control approaches which are inevitable when using computer science approaches.

Having said that, this research is still in its preliminary stage with the control design just implemented at the local server level and very far from its real implementation in actual data centres and needs a lot of efforts to reach the final stages. Due to time limitations, controller design for power optimization at the data centre level could not be studied deeply enough to propose a solution, and the integration of this work with previous ones had to be deferred.

Therefore, the future scope of this research work can include both development at the local level and the use of different strategies, such as model predictive control, H-infinity approaches and optimal control techniques, for the overall energy optimization of the data centres, and then its testing and implementation on real systems.

Bibliography

[1] Daniele Mastrandrea, *Object-oriented modelling and simulation of airflow in data centres based on a quasi-3D approach for energy optimisation*, MSc thesis, Politecnico di Milano, 2011.

[2] *Data centres energy consumption trends.* U.S. Department of Energy, 2010.

[3] Patrick Thibodeau. *Data centres are the new polluters.* http://www.computerworld.com/article/2598562/data-centre/data-centres-are-the-new-polluters.html.

[4] C. H. Hsu. *Compiler-Directed Dynamic Voltage and Frequency Scaling for CPU Power and Energy Reduction.* Ph. D. Dissertation, the State University of New Jersey, USA, 2003.

[5] T. D. Bord *Energy-Efficient Processor System Design.* Ph. D. Dissertation, University of California, Berkeley, USA, 2001.

[6] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. *Reducing Power in High-performance Microprocessors.* In Proceedings of the 35th Conference on Design Automation, ACM, USA, June 1998.

[7] Suleiman, D., M. Ibrahim, and I. Hamarash. *Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction.* 4th International Conference on Electrical and Electronics Engineering,2005.

[8] Meisner, David,Brian T. Gold, and Thomas F.Wenisch. *PowerNap: eliminating server idle power.*.ACM Sigplan Notices. Vol. 44. No. 3. ACM, 2009.

[9] Beloglazov, Anton,and Rajkumar Buyya. *Energy efficient allocation of virtual machines in cloud data centres.*Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on (pp. 577-578). IEEE, 2010.

[10] Cochran, Ryan, et al. *Pack and Cap:adaptive DVFS and thread packing under power caps.*Proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture. ACM, 2011.

[11] Leverich, Jacob, et al. *Power management of datacentre workloads using per-core power gating.*.Computer Architecture Letters 8.2 (2009): 48-51.

[12] Nathuji, Ripal, Canturk Isci, and Eugene Gorbatov. *Exploiting platform heterogeneity for power efficient data centres.*Autonomic Computing, 2007. ICAC'07. Fourth International Conference on. IEEE, 2007.

[13] Wang, Lizhe, et al. *Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS.* Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on. IEEE, 2010.

[14] Rambo, J., and Joshi, Y. *Modelling of data centre airflow and heat transfer: State of the art and future trends.* Distributed and Parallel Databases, 21(2-3), 193-225, 2007.

[15] Bash, C. E., Patel, C. D., and Sharma, R. K. *Efficient thermal management of data centres—Immediate and long-term research needs.* HVAC and R Research, 9(2), 137-152, 2003.

[16] Huang, Wei, et al. *Tapo: Thermal-aware power optimization techniques for servers and data centres.* Green Computing Conference and Workshops (IGCC), 2011 International. IEEE, 2011.

[17] Jeandel A., Boudaud F. *Physical System Modelling Languages: from ALLAN to Modelica.* Building Simulation'97, IBPSA Conference, Prague, September 8–10, 1997.

[18] Leva, Alberto, and Alessandro Vittorio Papadopoulos. *Tuning of event-based industrial controllers with simple stability guarantees.* Journal of Process Control 23.9 (2013): 1251-1260.

[19] Pelley, Steven, et al. *Understanding and abstracting total data centre power.* Workshop on Energy-Efficient Design. 2009.