POLITECNICO DI MILANO

Corso di Laurea Specialistica in Ingegneria Informatica

Dipartimento di Elettronica e Informazione

OBJECT-ORIENTED MODELLING AND SIMULATION OF AIR FLOW IN DATA CENTRES BASED ON A QUASI-3D APPROACH FOR ENERGY OPTIMISATION

Relatore:

Prof. Alberto Leva

Correlatore:

Ing. Marco Bonvini

Tesi di:

Daniele Mastrandrea, 739318

Anno Accademico 2010 - 2011

. . . this page intentionally left blank . . .

Contents

List of Figures

List of Code Snippets

Sommario

Il sempre crescente impiego nella vita di tutti i giorni di soluzioni nell'ambito delle telecomunicazioni e dell'*information technology* porta con sè una parallela evoluzione delle infrastrutture a supporto.
Inizialmente i server Internet erano collocati in uffici tradizionali senza alcuna idea di centralizzazione. Con l'aumentare delle necessità da parte dei server di alimentazione senza interruzione e di condizionamento dell'aria di precisione, a causa dell'importante mole di calore dissipato dalla componentistica utilizzata, sono stati costruiti centri di elaborazione dati (CED) – spesso indicati col termine inglese *data centre* – in modo da concentrare tutte le risorse di elaborazione in un'unica stanza che ne potesse soddisfare appieno le stringenti necessità.
La distribuzione di calore all'interno della stanza può variare in maniera significativa da una zona all'altra, e cambia con l'aggiunta di nuovo hardware o la riconfigurazione di parte di quello già esistente. Perciò, per evitare la rottura di costosi macchinari diventa necessario prestare particolare attenzione a come l'aria calda e l'aria fredda si distribuiscono nel locale macchine. É evidente come a tal fine non sia più sufficiente dimensionare l'apparato di raffreddamento in modo tale che la sua capacità refrigerante sia almeno pari alla potenza termica immessa nella stanza dai server.
La *fluidodinamica computazionale* – o in inglese *Computational Fluid Dynamics*, CFD – fornisce un valido strumento a supporto della modellistica di un data centre, in modo da studiare l'impatto di un layout interno alla stanza rispetto a un altro. Soluzioni basate su questo tipo di approccio sono senz'altro da preferirsi

ai tentativi sul campo, poichè le simulazioni necessarie alle verifiche del caso sono molto più rapide ed economiche rispetto all'implementazione fisica di una disposizione della stanza.
Lo scopo di questo lavoro è quello di produrre una libreria che renda possibile la simulazione del comportamento di un data centre utilizzando un approccio simile a quello impiegato dai sistemi CFD. Tuttavia, a differenza di quanto accade per una tipica simulazione CFD statica, che può durare anche 24 ore o più, l'efficienza dei modelli sviluppati è tale da consentire di portare a termine lunghe simulazioni statiche in tempi dell'ordine dei minuti, o simulazioni dinamiche in tempistiche paragonabili a metà del *real time*; a titolo d'esempio, con un orizzonte di simulazione di due ore i risultati possono essere ottenuti in meno di un'ora.
Purtroppo il miglioramento di un aspetto porta al peggioramento di un altro: le assunzioni fatte per consentire rapidi tempi di simulazione implicano che i modelli sviluppati penalizzino la perfetta rappresentazione del campo delle velocità all'interno della stanza: pur ottenendo velocità consistenti con quelle misurabili, il campo di moto può differire in maniera a volte non trascurabile. Ciononostante, la distribuzione delle temperature non è, in media, influenzata dal compromesso appena descritto.
Quest'ultima proprietà rende possibile la corretta stima degli scambi termici tra gli elementi della stanza e quindi di confrontare l'impatto sull'efficienza energetica di qualche cambiamento effettuato nel data centre.
Una volta ottenuto un modello affidabile del comportamento termico della stanza, sarà possibile impiegarlo per confrontare l'efficacia di alcune delle strategie di controllo disponibili e stabilire quale sia la migliore in termini energetici.
L'architettura di controllo utilizzata come riferimento è *a due livelli*, in cui il livello superiore si occupa di tenere traccia delle richieste ricevute dall'esterno e di consegnarle al livello inferiore che opera da *bilanciatore di carico*, decidendo quali server tenere accesi e come ripartire le richieste da elaborare. Al livello superiore compete anche l'*extrema ratio* di spegnere un server che dovesse andare in uno stato di protezione termica, ovvero qualora la temperatura al suo ingresso dovesse

superare un valore di soglia in genere stabilito dal produttore.
Una decisione intermedia che il controllore di livello superiore può prendere qualora si raggiungesse una temperatura critica – in generale *non* stabilito dal produttore dei macchinari, ma comunque inferiore alla temperatura massima di funzionamento – è quella di scalare la frequenza del server in questione o di ridurne momentaneamente il carico, in modo tale da impedire al server di contribuire ulteriormente all'immissione di aria calda che può potenzialmente ricircolare fino al proprio ingresso. La frazione di carico così rifiutata non deve andare persa, ma spetta ancora al controllore di livello superiore il compito di riaccodare ciò che altrimenti andrebbe abbandonato.
Il bilanciatore di carico è l'oggetto il cui comportamento può essere il più variegato. Lo scopo ultimo del lavoro, dunque, è quello di fornire uno strumento valido e affidabile per confrontare l'efficienza energetica di alcune strategie di bilanciamento. A tal fine sono stati realizzati, a titolo esemplificativo e assolutamente non esaustivo, due controllori di livello inferiore. Il primo implementa la strategia più semplice in assoluto: mantiene accesi tutti i server e ripartisce il carico in maniera uniforme su tutte le macchine. La seconda strategia è molto più sofisticata.
A ogni ripetizione di un intervallo di campionamento, il bilanciatore calcola il numero minimo di server necessari a soddisfare il numero medio di richieste ricevute durante il precedente intervallo e sceglie quali accendere in base allo stato precedente e alla sua temperatura in ingresso. Viene dunque effettuato un primo passo di allocazione in cui solo le macchine già accese allo step precedente vengono considerate, dalla più fredda alla più calda; il secondo passo, se necessario, accende server in precedenza inattivi seguendo la stessa logica. La nuova allocazione diviene effettiva solo dopo il passaggio di un intervallo temporale che rappresenta il tempo necessario a un server per effettuare le procedure di avvio. Il carico viene, invece, ripartito continuamente dandone la quantità massima al più freddo server disponibile, e così via fino all'esaurimento del carico di lavoro.
Le ragioni alla base della seconda strategia proposta sono molteplici. Una su tutte è la minimizzazione delle risorse impiegate, ottenuta con l'accensione del

numero minimo di macchine. L'altra idea alla base di tale politica di allocazione è quella di minimizzare il numero di accensioni e spegnimenti a cui una macchina è soggetta. Infatti, il tempo di vita dei dischi fissi di macchine di fascia server può risultare fortemente ridotto qualora questi fossero soggetti a ripetuti cicli di acceso-spento. Unitamente a ciò, l'idea alla base della strategia è quella di minimizzare i consumi energetici.
É importante ribadire che, nonostante le simulazioni riportino valori numerici di velocità, questi valori non riflettano necessariamente le velocità realmente misurabili all'interno della stanza.

Chapter 1

Introduction

As the utilisation of communication and information technology grows in every-day use, a parallel evolution of the nature of the supporting infrastructure is unavoidable.
When the Internet started to spread, servers were located in traditional offices, a solution not involving any kind of centralisation. With the increasing need for reliable power supplies (i.e., UPS[1]) along with the need for climate control systems due to the significant amount of heat dissipated by the machinery, Internet data centres were built in *ad hoc* facilities for concentrating machines in one room able to meet all the requirements.
Heat distribution can vary significantly across the computer room, and the addition or reconfiguration of hardware just remarks such variation. Therefore, to prevent equipment failure, just meeting the total cooling capacity or airflow requirement is not enough; special attention must be paid to the distribution of the cooling air. Moreover, the data centre design cannot rely on intuitive distribution of air. The fluid mechanics and heat transfer processes inside the data centre must be carefully considered. Hence the need to model the air flow and temperature distribution in the data centre.
Computational Fluid Dynamics (CFD) modelling offers a practical and compre-

[1]Uninterruptible Power Supply

hensive design approach. Computational simulations can be used for a quick setup of any proposed layout, any desired placement of conditioning units, and any imagined failure scenario.
The "computational" trial-and-error process is widely preferable, since performing a simulation is much faster and more economical than building an actual layout.

1.1 Motivation

The final outcome of this work is a framework that makes it possible to simulate the behaviour of a data centre using an approach similar to that adopted in the CFD systems (e.g., Ansys Fluent).
However, unlike what happens in a typical static CFD simulation, which can even last 24 hours or more, the efficiency of the developed models is such to carry out static simulations in the order of minutes, while dynamic simulations can be obtained in time compared to halved real time; for example by setting a simulation horizon of 2 hours, results can be achieved in less than an hour.
Unfortunately, the improvement in one aspect leads to the deterioration of another: the developed models sacrifice the perfect representation of the velocities' field inside the room. Nevertheless, the temperature distribution is not, on average, affected by the trade-off just described.
This property makes it possible to correctly estimate the exchange of thermal power in the room – thus electrical power, knowing the coefficient of performance of the machines used – and then to compare the impact on energy efficiency of a few changes made in the room.
The ultimate aim of the work is therefore to provide a valid and reliable tool for comparing multiple control strategies and determine which is the best in energetic terms. It is important to reiterate that, despite simulations provide numerical values for velocities, these values do not reflect real values actually measurable in the room.

1.2 Related Works

Related works in this area include using CFD models to create intricate mathematical models of the data centre dynamics to find out optimal layouts or hot spots.
For instance, Patel et al. developed a CFD model in order to use inlet temperatures as the key measure of a proper data centre thermal design [18], while Karki et al. describe a CFD model for calculating airflow rates through perforated tiles in raised-floor data centres [15].
As a further step, Cremonesi and Sansottera employed a simplified model – yet computationally heavy – to devise a linear model based on the assumption that the fraction of air recirculating from one server to another is constant [9], [16]. The obtained model seems to perform fine, but the extensive use of CFD simulation during the training phase, combined with its intrinsically static nature, makes it sensitive to any kind of change: for a simple alteration – e.g., in the power consumption – a huge number of CFD simulations must be performed, thus leading to a considerable waste of time.

1.3 Organisation

The following chapters of the document are so structured:

- Chapter 2 introduces some concepts about modelling;
- Chapter 3 states the problem explaining what a data centre is and the related issues;
- Chapter 4 looms the ideas behind the models in the library;
- Chapter 5 shows the implementation of each model;
- Chapter 6 outlines some case studies of interest;
- Chapter 7 sums everything up.

Chapter 2

Modelling

The entire simulation process, when using a computer as a supporting tool, is divided into three main steps: mathematical modelling, implementation in some programming language, then the real simulation. This section will give a short overview about this process, describing every single step just mentioned.

2.1 Mathematical Modelling

A mathematical model usually depicts a system by a set of variables and a set of equations that establish relationships between the variables. The values of the variables can be practically anything; real to integer numbers, boolean values to strings. The variables represent some properties of the system, for example, measured system outputs often in the form of signals, timing data, counters, and event occurrence (yes/no). The actual model is the set of functions which describe the relations between the different variables.
Mathematical modelling problems are often classified into black box or white box models, according to how much *a priori* information is available of the system. A black box model is a system of which there is no *a priori* information available. A white box model (also called glass box or clear box) is a system where all necessary information is available. Practically all systems are somewhere between the black box and white box models, so this concept only works as an intuitive

guide for approach. Usually it is preferable to use as much *a priori* information as possible to make the model more accurate. Therefore white box models are usually considered easier, because if information is correctly used, then the model will behave correctly. Often the *a priori* information comes in forms of knowing the type of functions relating different variables. In black box models one tries to estimate both the functional form of relations between variables and the numerical parameters in those functions. Using *a priori* information the model could end up, for example, with a set of functions that probably could describe the system adequately. If there is no *a priori* information one would try to use functions as general as possible to cover all different models. An often used approach for black box models is the usage of neural networks which usually do not make assumptions about incoming data. The problem with using a large set of functions to describe a system is that estimating the parameters becomes increasingly difficult when the amount of parameters (and different types of functions) increases. Any model which is not pure white box contains some parameters that can be used to fit the model to the system it shall describe.
To provide an example for the whole modelling process, here is shown how a model for a simple RC network can be obtained. Figure 2.1 depicts the electrical scheme of the circuit to model.

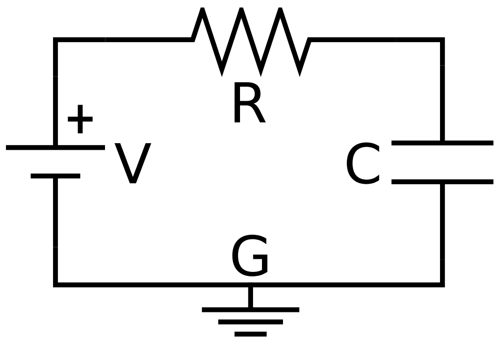


Figure 2.1: Simple Resistor-Capacitor network

Applying Kirchhoff's Voltage Law (KVL), the equation governing the circuit is:

$$V = Ri(t) + v_{\mathrm{c}}(t) \tag{2.1}$$

where $i(t)$ is circulating current, $v_\mathrm{c}(t)$ is the tension on the capacitor, V is the tension, assumed to be constant, on the generator.
Substituting the characteristic equation of the capacitor, the linear differential equation obtained is:

$$V = RC\frac{dv_\mathrm{c}(t)}{dt} + v_\mathrm{c}(t) \quad \rightarrow \quad \frac{dv_\mathrm{c}(t)}{dt} + \frac{1}{RC}v_\mathrm{c}(t) - \frac{V}{RC} = 0 \tag{2.2}$$

calling $\tau = RC$ time constant, and solving equation 2.2 with the initial condition $v_\mathrm{c}(0) = v_0$, here is the resulting solution:

$$v_\mathrm{c}(t) = (v_\mathrm{c}(0) - V)e^{-\frac{t}{\tau}} + V \tag{2.3}$$

The model here obtained is a blend between a white and a black box model, or most likely a parametric white box model.
In order to get a clear box model, any of the parameters should be specified, for example: $V = 1\,\mathrm{V}$, $R = 2.2\,\mathrm{k\Omega}$, $C = 1\,\mathrm{\mu F}$ and $v_0 = 0\,\mathrm{V}$. A completely specified model is then obtained:

$$v_\mathrm{c}(t) = (0 - 1)e^{-\frac{t}{(2.2\times10^3)(1\times10^{-6})}} + 1 = 1 - e^{-2200t} \tag{2.4}$$

2.2 Implementation

Once model's equations are written, next step is to implement them in some programming language. One of the most convenient language available at the moment is Modelica, which is an object-oriented, multi-domain modelling language for component-oriented modelling of complex systems (e.g. systems containing mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents). The free Modelica language is developed by the non-profit Modelica Association. While Modelica resembles object-oriented programming languages, such as C/C++ or Java, it differs in two important aspects:

- Modelica is a modelling language rather than a true programming language. Modelica classes are not compiled, in the usual sense, but they are translated into objects which are then ran by a simulation engine. The simulation engine is not specified by the language, even if some required capabilities are outlined;

- Although classes may contain algorithmic components similar to statements or blocks in programming languages, their primary content is a set of equations. In contrast to a typical assignment statement, such as $x := 2 - y$, where the left-hand side of the statement is assigned a value calculated from the expression on the right-hand side, an equation may have expressions on both its right- and left-hand sides, for example $x + y = 2$. Equations do not describe assignment but equality: in Modelica terms, equations have no pre-defined causality. The simulation engine may (and usually must) manipulate the equations symbolically to determine their order of execution and which components in the equation are inputs and which are outputs.

To go on with the previous example, Modelica code looks as follows:

RC.mo

```
model RC
  parameter Modelica.SIunits.Voltage V=1;
  parameter Modelica.SIunits.Resistance R=2.2e3;
  parameter Modelica.SIunits.Capacitance C=1e-6;
  parameter Modelica.SIunits.Voltage v_0=0;
  Modelica.Electrical.Analog.Sources.ConstantVoltage source(V=V);
  Modelica.Electrical.Analog.Basic.Resistor resistor(R=R);
  Modelica.Electrical.Analog.Basic.Capacitor capacitor(C=C);
  Modelica.Electrical.Analog.Basic.Ground ground;
equation
  connect(sineVoltage.p, resistor.p);
  connect(resistor.n, capacitor.p);
  connect(capacitor.n, ground.p);
  connect(ground.p, sineVoltage.n);
initial equation
  capacitor.v = v_0;
end RC;
```

Code Snippet 1: RC network - Modelica text model

Modelica language also provides a simple notation to provide graphical visualisation, then the model just written looks like in figure 2.2.

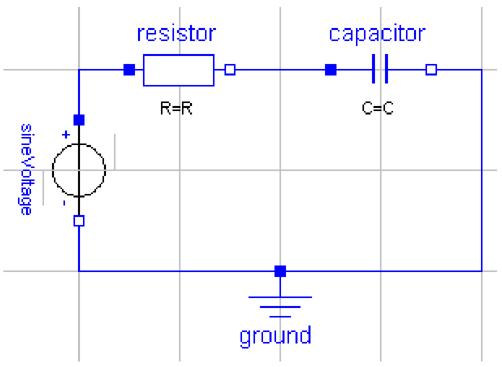


Figure 2.2: RC network - Modelica graphical model.

2.3 Simulation

The final simulation, with the plot of the results, is done translating Modelica code into C/C++ (to reach maximum speed), which is then compiled and executed, resulting in the building of a result file. The entire process can be software aided by some tools, like OMEdit, which is the one developed by the Open Source Modelica Consortium (OSMC); OMEdit and its open source translator would be the best free choice for the modelling of electrical systems, but all the plots and screenshots in these pages are obtained using a non-free tool because it supports some features not yet ported into OpenModelica.
Dymola, by the Swedish company Dynasim AB (now part of Dassault Systems), is one of the commercial software for Modelica development. This tool is rather similar to an IDE, because it provides both a text and a graphic editor, and an interface to build the source file, then to plot the results or to debug any error. Simulation's outcome is a `.mat` file, which contains *each* value for *each* of the variables at *each* integration step time. Dymola is available for both Linux and Windows systems. Figure 2.3 shows the graphics obtained in Dymola after the simulation of the previously modelled circuit.

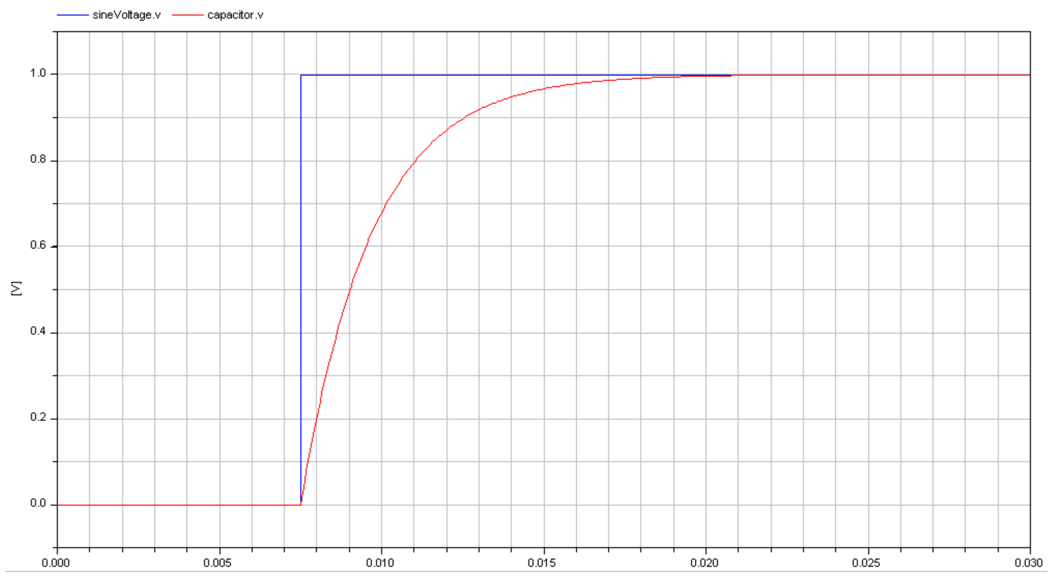


Figure 2.3: RC network plot in Dymola.

Notice that the model has been slightly modified in order to achieve these results: the voltage source is a step at the time of 0.075 s to show the transient of the circuit after having such a signal as an input.

Chapter 3

Problem statement

As Albert Einstein stated, «The formulation of a problem is often more essential than its solution, which may be merely a matter of mathematical or experimental skill».
This chapter will introduce some concepts regarding data centres in order to make said solution much easier to devise. A clear problem statement should also help in being self confident that the found solution is correct.
Some care should be taken, though, since «For every complex problem there is an answer that is clear, simple, and wrong», as mentioned by Henry Louis Mencken.

3.1 Data centre

A data centre is a facility used to house computer systems and associated components, such as telecommunications and storage systems. It generally includes redundant or backup power supplis.
Data centres have their roots in the huge computer rooms of the early ages of the computing industry. Early computer systems were complex to operate and maintain, and required a special environment in which to operate. Many cables were necessary to connect all the components, and methods to accommodate and organise them were devised, such as standard rack mount equipment, elevated floors and cable trays (installed overhead or under the elevated floor).

Also, a single mainframe required a great deal of power, and had to be cooled to avoid overheating. Security was important since computers were expensive, and were often used for military purposes.

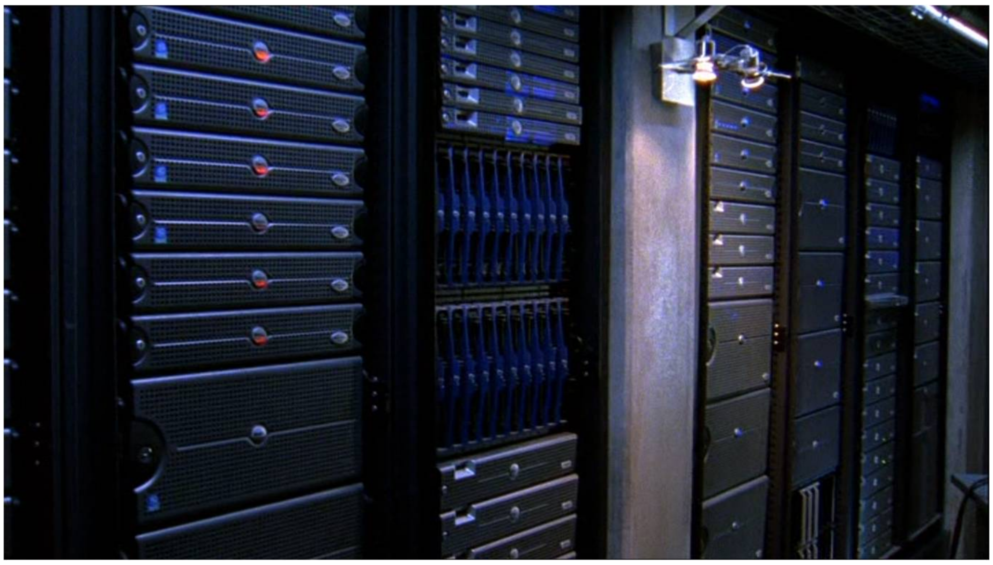


Figure 3.1: The DU 97 Supercomputer featured in the spy-drama TV series *Chuck*

During the boom of the microcomputer industry, and especially during the 1980s, computers started to be deployed everywhere, in many cases with little or no care about operating requirements. However, as Information Technology operations started to grow in complexity, companies grew aware of the need to control IT resources. With the advent of client-server computing, during the 1990s, microcomputers (now called “servers”) started to find their places in the old computer rooms. The availability of inexpensive networking equipment, coupled with new standards for network cabling, made it possible to use a hierarchical design that put the servers in a specific room inside the company. The use of the term “data centre”, as applied to specially designed computer rooms, started to gain popular recognition about this time.

Nowadays, data centre design, construction, and operation is a well-known discipline. Standard Documents from accredited professional groups, such as the Telecommunications Industry Association, specify the requirements for data centre design. Well-known operational metrics for data centre availability can be used to evaluate the business impact of a disruption. There is still a lot of development being done in operation practice, and also in environmentally-friendly data centre design. Data centres are typically very expensive to build and maintain. For instance, Facebook's 28 000 m^2 brand new data centre in Prineville, Oregon, was initially valued at \$200 million, but the expense has doubled in scale since then and may be even greater [19].

Figure 3.2: A typical server room

IT operations are a crucial aspect of most organisational operations. One of the main concerns is business continuity; companies rely on their information systems to run their operations. If a system becomes unavailable, company operations may be impaired or stopped completely. It is necessary to provide a reliable infrastructure for IT operations, in order to minimise any chance of fault. A data centre must therefore keep high standards for assuring the integrity and functionality of its hosted computer environment. This is accomplished through redundancy of both fibre optic cables and power, which includes emergency backup power generation.

Standardisation and modularity can yield savings and efficiencies in the design and construction of telecommunications data centres. There is a trend to modernise data centres in order to take advantage of the performance and energy efficiency increases of newer IT equipment and capabilities, such as cloud computing. This process is also known as data centre transformation.
Data centre transformation takes a step-by-step approach through integrated projects carried out over time. This differs from a traditional method of data centre upgrades that takes a serial approach. The typical projects within a data centre transformation initiative include standardisation/consolidation, virtualisation, automation and security.

Standardisation/consolidation The purpose of this project is to reduce the number of data centres a large organization may have. This project also helps to reduce the number of hardware, software platforms, tools and processes within a data centre. Organisations replace ageing data centre equipment with newer ones that provide increased capacity and performance. Computing, networking and management platforms are standardised so they are easier to manage;

Virtualise There is a trend to use IT virtualisation technologies to replace or consolidate multiple data centre equipment, such as servers. Virtualisation helps to lower capital and operational expenses, and reduce energy consumption;

Automating Data centre automation involves automating tasks such as provisioning, configuration, patching, release management and compliance. As enterprises suffer from few skilled IT workers, automating tasks make data centres run more efficiently;

Securing In modern data centres, the security of data on virtual systems is integrated with existing security of physical infrastructures. The security of a modern data centre must take into account physical security, network security, and data and user security.

A data centre can occupy one room of a building, one or more floors, or an entire building. Most of the equipment is often in the form of servers mounted in 19 inch rack cabinets, which are usually placed in single rows forming corridors (so-called *aisles*) between them. This allows people access to the front and rear of each cabinet.

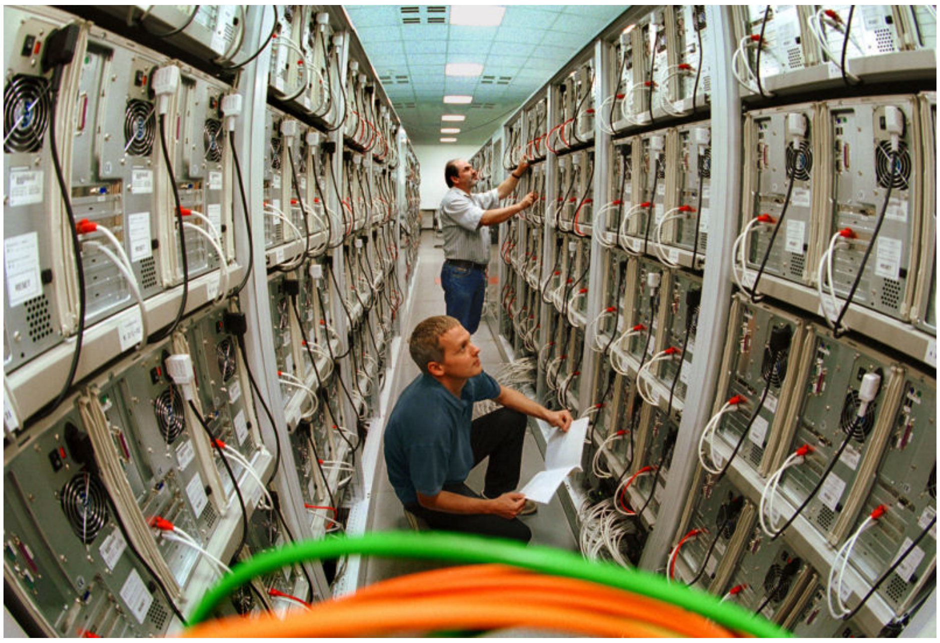


Figure 3.3: Cold aisle between two rows of servers

Servers differ greatly in size from 1 U[1] servers to large freestanding storage silos which occupy many tiles on the floor. Some equipment such as mainframe computers and storage devices are often as big as the racks themselves, and are placed alongside them. Very large data centres may use shipping containers packed with thousands or more servers each; when repairs or upgrades are needed, whole containers are replaced rather than repairing individual servers.
The physical environment of a data centre is rigorously controlled. Air conditioning

[1]A Rack Unit or U (less commonly RU) is a unit of measure used to describe the height of equipment intended for mounting in a rack. 1 U = 1.75 inch = 44.45 mm.

is used to control the temperature and humidity in the data centre. ASHRAE's "Thermal Guidelines for Data Processing Environments" recommend a temperature range of 16 °C to 24 °C and humidity range of 40 % to 55 % with a maximum dew point of 15 °C as optimal for data centre conditions. The temperature in a data centre will naturally rise because the electrical power used heats the air. Unless the heat is removed, the room temperature will rise, resulting in electronic equipment malfunction. By controlling the air temperature, the server components at the board level are kept within the manufacturer's specified temperature/humidity range.

Data centres house IT equipment recommended (by IT vendors) to operate within certain temperature ranges. Often today, with the ever growing information processing densities, data centres servers continually run the risk of producing heat that goes well beyond these guidelines, even when using precision cooling units. Overheating of data centre equipment can result in:

- Reduced server performance or total non functionality due to recirculated, hot exhaust air from IT equipment finding its way into an inlet;
- Precision cooling equipment being set at temperatures lower than recommended due to air stratification or the layering of different temperature air masses in the data centre;

Air flow management strategies can best be applied through hot aisle containment measures. Containment of hot or cold aisles or ducting hot air from cabinets is intended to prevent air temperatures being mixed within server rooms. Generally rows of cabinets will be placed facing each other so that cooling air going to equipment cabinets can reach the equipment air intakes at the temperature set point for the room. When cold intake air is separated from hot exhaust air, cooling equipment can operate more efficiently and can be safely set to use higher temperatures with less dehumidification than if cold and hot air was being mixed within a server room.

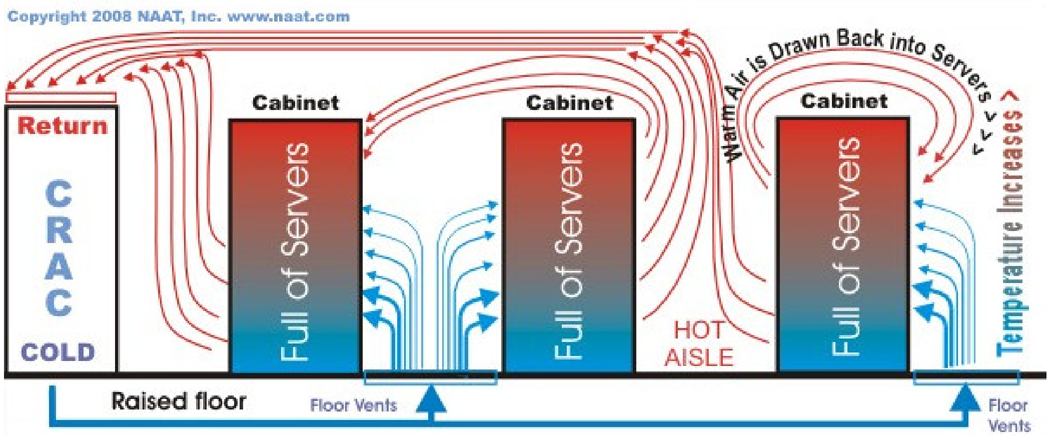


Figure 3.4: Air recirculation and raised floor

Energy use is a central issue for data centres. Power draw for data centres ranges 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. For higher power density facilities, electricity costs are a dominant operating expense and account for over 10 % of the total cost of ownership (TCO) of a data centre. By 2012 the cost of power for the data centre is expected to exceed the cost of the original capital investment.

The main purpose of a data centre is running the applications that handle the core business and operational data of the organization. Such systems may be proprietary and developed internally by the organisation, or bought from enterprise software vendors. A data centre may be concerned with just operations architecture or it may provide other services as well. Often these applications will be composed of multiple hosts, each running a single component. Common components of such applications are databases, file servers, application servers, middleware, and various others.

3.2 Server

A server is a physical computer dedicated to running one or more such services (as a host), to serve the needs of 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 other. Servers provide essential services across a network, either to private users inside a large organization or to public users via the Internet.

In the hardware sense, the word “server” typically designates computer models intended for 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 other. 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, increased high-performance RAM, and typically more than one large hard drive. More obvious distinctions include marked redundancy in power supplies, network connections, and even the servers themselves.

Hardware requirements for servers vary, depending on the server application. Absolute CPU speed is not usually as critical to a server as it is to a desktop machine. Servers' duties to provide service to many users over a network lead to different requirements such as fast network connections and high I/O throughput. Since servers are usually accessed over a network, they may run in headless mode without a monitor or input device. Processes that are not needed for the server's function are not used. Many servers do not have a graphical user interface (GUI) as it is unnecessary and consumes resources that could be allocated elsewhere. Similarly, audio and USB interfaces may be omitted.

Servers often run for long periods without interruption and availability must often be very high, making hardware reliability and durability extremely important. Although servers can be built from commodity computer parts, mission-critical enterprise servers are ideally very fault tolerant and use specialised hardware with low failure rates in order to maximise uptime, for even a short-term failure can cost more than purchasing and installing the system. Servers may incorporate faster, higher-capacity hard drives, larger computer fans or water cooling to help remove

Figure 3.5: An example of servers without any graphical interface

heat, and UPS units that ensure the servers continue to function in the event of a power failure. These components offer higher performance and reliability at a correspondingly higher price. Hardware redundancy – installing more than one instance of modules such as power supplies and hard disks arranged so that if one fails another is automatically available – is widely used.

To increase reliability, most of the servers use memory with error detection and correction, redundant disks, redundant power supplies and so on. Such components are also frequently hot swappable, allowing technicians to replace them on the running server without shutting it down. To prevent overheating, servers often have more powerful fans.

As servers need stable power supply, good Internet access, increased security and are also noisy, it is usual to store them in dedicated server centres or special rooms. This requires to reduce power consumption as extra energy used generates more heat and the temperature in the room could exceed the acceptable limits. Normally server rooms are equipped with air conditioning devices. Server casings

are usually flat and wide, adapted to store many devices next to each other in server rack. Unlike ordinary computers, servers usually can be configured, powered up and down or rebooted remotely, using out-of-band management.
Many servers take a long time for the hardware to start up and load the operating system. Servers often do extensive pre-boot memory testing and verification and startup of remote management services. The hard drive controllers then start up banks of drives sequentially, rather than all at once, so as not to overload the power supply with startup surges, and afterwards they initiate RAID system pre-checks for correct operation of redundancy. It is common for a machine to take several minutes to start up, but it may not need restarting for months or years.

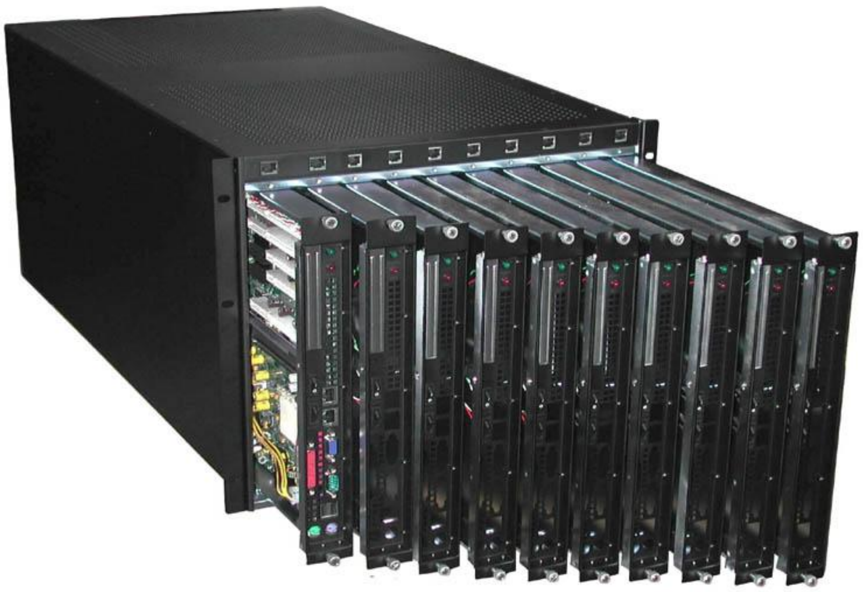


Figure 3.6: Full-height blades

A *blade server* is a stripped down server computer with a modular design optimised to minimise the use of physical space and energy. Whereas a standard rack-mount server can function with (at least) a power cord and network cable, blade servers have many components removed to save space, minimise power consumption and other considerations, while still having all the functional components to be considered a computer.

A *blade enclosure*, which can hold multiple blade servers, provides services such as power, cooling, networking, various interconnects and management. Together, blades and the blade enclosure form the blade system.

In a standard server-rack configuration, 1 U defines the minimum possible size of any equipment. The principal benefit and justification of blade computing relates to lifting this restriction so as to reduce size requirements. The most common computer rack form-factor is 42 U high, which limits the number of discrete computer devices directly mountable in a rack to 42 components. Blades do not have this limitation.

The enclosure (or chassis) performs many of the non-core computing services found in most computers. Non-blade systems typically use bulky, hot and space-inefficient components, and may duplicate these across many computers that may or may not perform at capacity. By locating these services in one place and sharing them between the blade computers, the overall utilisation becomes more efficient. The specifics of which services are provided may vary by vendor.

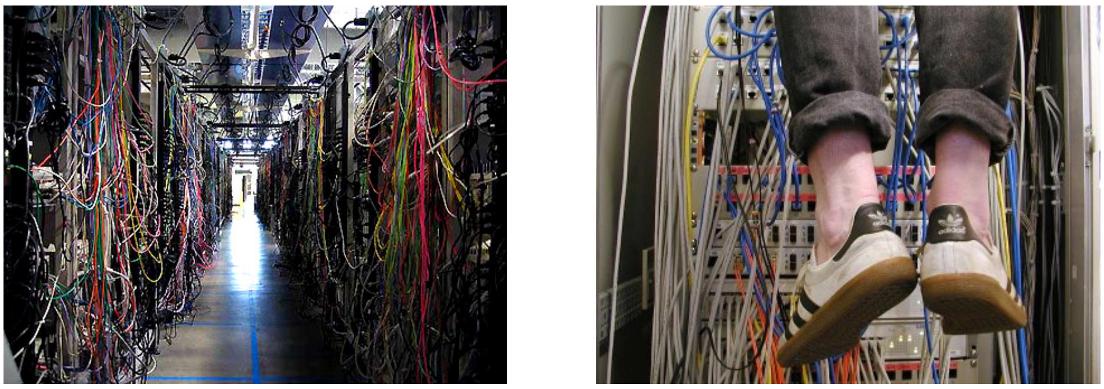

(a) Tangled cables **(b)** The results of trying to move a server

Figure 3.7: Spaghetti cabling

Blade approach may help in preventing the phenomenon known as “Spaghetti cabling”, depicted in figure 3.7

Blade servers function well for specific purposes such as web hosting, virtualisation, and cluster computing. Individual blades are typically hot-swappable. As users add more processing power, memory and I/O bandwidth to blade servers, they deal with larger and more diverse workloads.

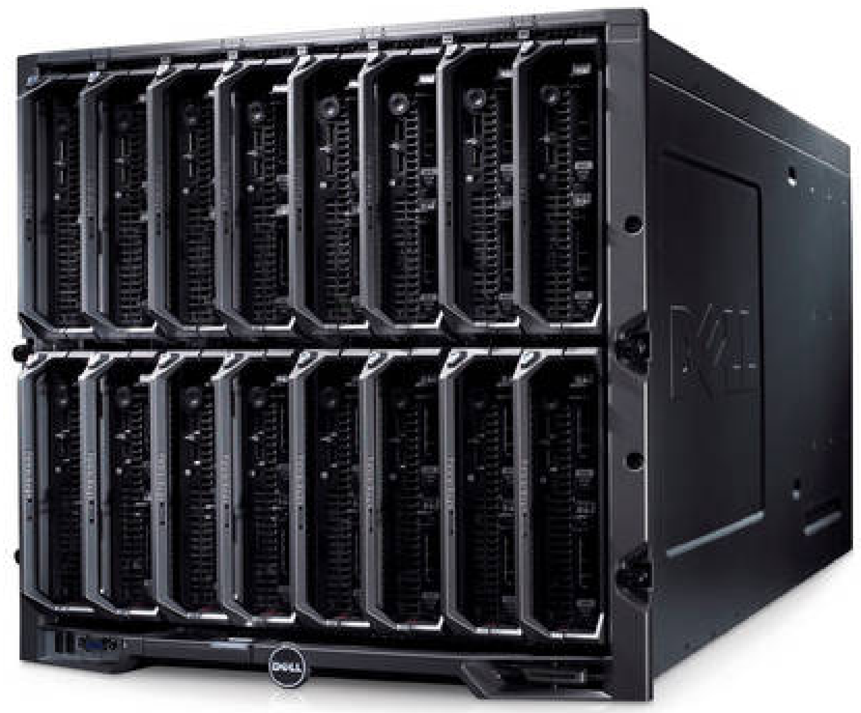


Figure 3.8: Dell PowerEdge M1000e blade system, fully populated by half-height blades

Blade servers do not, however, provide the answer to every computing problem. Very large computing tasks may still require server farms of blade servers, and because of blade servers' high power density, can suffer even more acutely from the HVAC[2] problems that affect large conventional server farms. According to Industrial Market Trends, racks of servers can get as hot as a seven-foot tower of powered toaster ovens.

[2]Heating, Ventilation and Air Conditioning

3.3 CRAC

A Computer Room Air Conditioning (CRAC) unit, sometimes replaced by a Computer Room Air Handler (CRAH) unit, is a device that monitors and maintains the temperature, air distribution and humidity in a network room or data centre. CRAC units are replacing air-conditioning units that were used in the past to cool data centres.

There are a variety of ways that the CRAC units can be situated. One CRAC setup that has been successful is the process of cooling air and having it dispensed through an elevated floor as in figure 3.4. The air rises through the perforated sections, forming cold aisles. The cold air flows through the racks where it picks up heat before exiting from the rear of the racks. The warm exit air forms hot aisles behind the racks, and the hot air returns to the CRAC intakes, which are positioned above the floor.

The performance of vapour compression refrigeration cycles is limited by thermodynamics. Air conditioning devices move heat rather than convert it from one form to another, so thermal efficiencies do not appropriately describe the performance of these devices.

The Coefficient Of Performance (COP) of a cooling machine is the ratio of heat removed from the cold reservoir to input work. COP is often non constant with inlet and outlet temperature. That is, the same ΔT will often result in different performance if the inlet temperatures are different.

The Energy Efficiency Ratio (EER) of a particular cooling device is the ratio of output cooling (in Btu/h) to input electrical power (in W) at a given operating point. It is related to the COP, with the primary difference being that the COP of a cooling device is unit-less. EER is calculated by multiplying COP by 3.412 which is the conversion factor from Btu/h to W.

3.4 Work's aim

According to this not-so-short introduction, the power consumption of a data centre can be roughly split into two parts: the power drawn by the servers and by the conditioning units.

3.4.1 Server power

A data centre is usually oversized with respect to the average workload it is supposed to delivery. This is the result of the sizing for the worst case scenario, namely the maximum number of expected jobs per time unit. Even when idle, a server still draws power for it being on; in this state it can react to an increment in the number of jobs and readily deliver them. On the counter, a server shut down won't consume any power, but it will not be immediately available upon computation capacity demand. Also, repeated power on/off may significantly reduce the life of the components, especially for what concerns hard drives.
What is needed in this situation is some kind of a trade-off between computational and electrical power. A smart policy may address this issue turning on a number of servers that may deliver an average load, reserving the right of booting some other ones upon need. The chosen servers should be those whose inlet temperature is the lowest (thus leading to low outlet temperature), giving priority to those that were already on. This should decrease the number of power on, and at the same time it should keep the data centre's activity into the standards.

3.4.2 CRAC power

A typical issue in the use of HVAC devices is that their efficiency strongly depends on the operating point. Usually, the lowest inlet temperature leads to the highest COP (or EER). The most straight way to achieve this is to lower the outlet temperature. Unfortunately, the COP also depends on the difference between inlet and outlet.

This situation is another one where some trade-off is required. The CRAC should:

- Keep servers' temperatures within operating ranges (mandatory);
- Keep its inlet and outlet temperature as low as possible;
- Keep its inlet and outlet temperature as close as possible;
- Keep its COP as high as possible.

Figure 3.9: Brand new (april 15th, 2011) Facebook's data centre in Prineville, Oregon. It received LEED (Leadership in Energy and Environmental Design) Gold Certification on November 17th, 2011, since it requires 52 % less energy to operate than a comparable facility built to code requirement

Chapter 4

Models

This chapter is about the concepts concerning the implemented models' background. The first part will deal with the subzonal models' framework, while the second will introduce the behaviour modelled by the models of the data centre's components. Finally, the third section will show some of the control strategies adopted.

4.1 Subzonal models

Dynamic modelling and simulation is very important for improving the energy efficiency of buildings. In that field, a primary challenge is to effectively confront the inherently multi-physic nature of the problem. The energy performance of a building results from heterogeneous phenomena (hydraulic, thermal, electric and so forth) and the operation of several control systems and the actions of the inhabitants. Worse still, energy performance is determined by the interaction of all those phenomena.

Virtually the totality of engineering tools and design practices broadly distinguish

- the building *stricto sensu*, i.e., walls, doors, windows and so on;
- the contained air volumes, possibly divided in zones;
- the HVAC system;

- automation and control systems;
- energy sources/sinks owing to the building utilisation, e.g., the heat released by occupants, industrial machines, or whatever is installed.

The subsystems' interaction is accounted for by having some of them provide boundary conditions for some others. This is quite far from an integrated approach, but tools that address the simulation of all (or at least part) of the subsystems in a coordinated way are at present little more than research objects.
The main reasons for such a scenario are the very different issues posed by the various subsystems. For example, control system models are made of oriented blocks and may need a continuous-time or a digital representation depending on the simulation purpose; HVAC models, conversely, live invariantly in the continuous-time domain, but are typically zero- or one-dimensional, while models of phenomena that occur in continua such as a wall or an air volume often cannot avoid three-dimensional distributions. As a result, it is difficult to devise models that address all the necessary phenomena, and can be organised in a modular way, to the advantage of their construction, parametrisation, and maintenance.
The work which this framework is based on concentrates on a major source of the mentioned problem, i.e., the modelling of air volumes. Specifically, it exploits the fact that whole buildings models are of interest for system level studies, where one can sacrifice detailed descriptions (e.g., of an air velocity field) in exchange for a correct representation of how all the phenomena interact to produce the main energy outcome: in some sense, although this is not the most correct definition from a theoretical standpoint, in system level simulation one is interested in realistically reproducing "what would be seen by a control and supervision system".
Based on the consideration above, an efficient compromise can be devised between models based on the simple and efficient (but often too imprecise) "fully mixed" paradigm, where air conditions in each volume are assumed uniform, and models based on the complex (but accurate) fine-scale three-dimensional CFD one.

The following sections will summarise the adopted methodology's final results, without lingering on details. For further information see [20], [5] and [6].

4.1.1 Balance equations

The mass, energy balance and the Navier Stokes (momentum) equations can be written as:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{4.1}$$

$$\frac{\partial (\rho e)}{\partial t} + \nabla \cdot (\rho \mathbf{v} h) = \nabla \cdot (k \nabla T) \tag{4.2}$$

$$\frac{\partial (\rho \mathbf{v})}{\partial t} + \nabla \cdot (\rho \mathbf{v} \mathbf{v}^{\mathrm{T}}) + \nabla p = \mathbf{f} \tag{4.3}$$

where the scalars p, T, e, h and ρ are respectively the fluid pressure, temperature, specific energy, specific enthalpy and density, the vectors $\mathbf{v}$ and $\mathbf{f}$ are the fluid velocity and the possible motion driving forces, and the scalar parameters k is the fluid thermal conductivity.

The finite-volume discretisation of the mass and energy equation is totally standard, thus not described for brevity. On the contrary, the spatial discretisation adopted for the momentum equation is novel, introducing an *ad hoc* approximation for the velocities' second derivatives, and a corresponding treatment of the boundary volume elements.

First the Newtonian fluid[1] simplification is adopted, then the convective term $(\rho \mathbf{v} \mathbf{v}^{\mathrm{T}})$ is here neglected, since its relevance is small when dealing with such small velocities as those of airflows in building rooms. These approximations lighten the model's nonlinearity, facilitating its numerical solution. Of course accuracy suffers in the description of highly local and directional phenomena like jets and plumes, but since the goal of the overall research is not to capture such details, said inaccuracy is largely acceptable. Also, only the component of air velocities

[1] A Newtonian fluid is a fluid whose stress versus strain rate curve is linear and passes through the origin. The constant of proportionality is known as the *viscosity*.

orthogonal to the volume surfaces are considered, leading *de facto* to a “quasi-3D” modelling approach.

Spatial discretisation is managed defining a staggered grid of points in the spatial domain of interest. Some of the nodes are attributed to control volumes, to which the mass and energy equations are referred. Some other nodes are conversely relative to the fluid motion among said volumes, thus being the locations to which velocities are attributed. In a parallelepiped-based grid the two sets of nodes form a “staggered” pattern, whence the name.

With reference to figure 4.1, blue elements – whose name is “Volume” – are the control volumes, while yellow ones – named “Layers” – are those accounted for velocities’ handling.

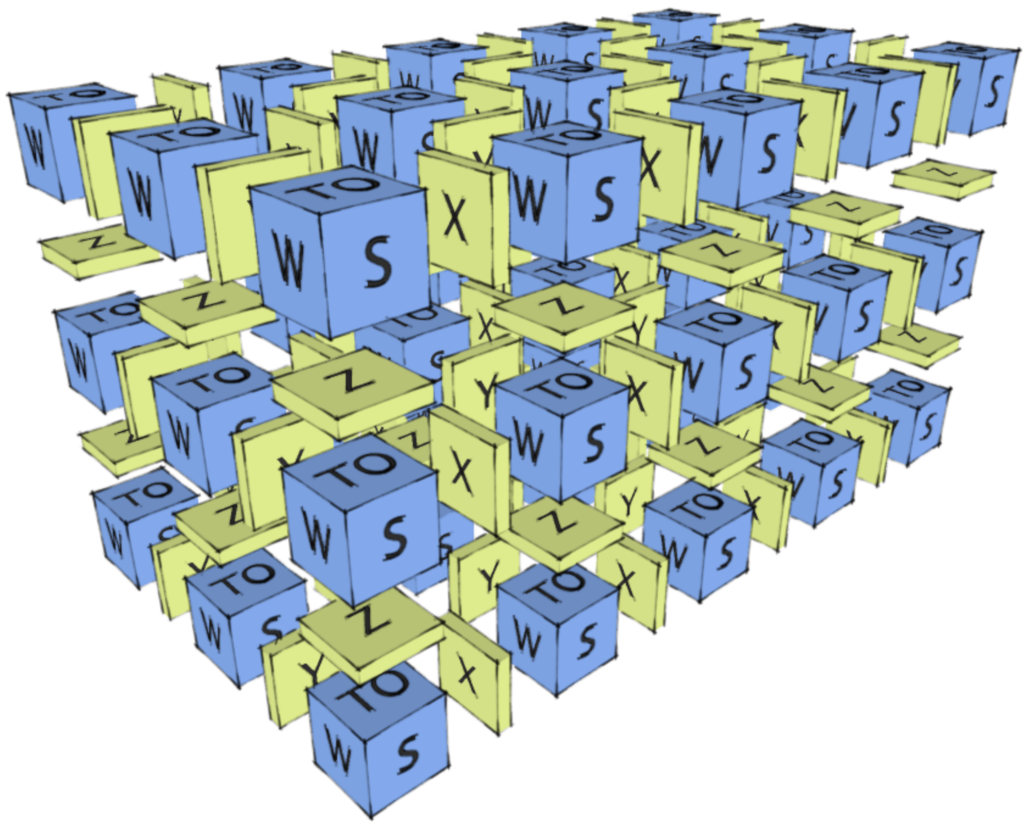


Figure 4.1: The staggered grid made up by Volumes and Layers

For the discretisation of the momentum equations, the velocity vector in each element is computed combining the components stored in the surrounding nodes.

Pressure is calculated similarly. The partial second derivative of the velocity is dealt adopting an *ad hoc* simplification, taking a second order polynomial function as a local approximant. Final discretised expression is obtained assuming constant viscosity.
The discretised momentum equations devised so far are valid in the volumes within a cavity (a room, a duct, a box, etc.) but apparently not for the volumes at the cavity boundaries, since velocity nodes referring to volumes at the boundary may not have one of the neighbours. A special momentum equation discretisation is thus required for boundary velocities.

4.1.2 Fluid state equations

The fluid considered here is air, treated as a mixture of ideal gases. Instead of using the ideal gas relationship, in order to simplify the model, the linearisation

$$\rho = \frac{p}{R^*T} \simeq \rho_o + \frac{1}{R^*T_o}P - \frac{p_o}{R^*T_o^2}(T - T_o) \tag{4.4}$$

is used here, where ρ is the fluid density, R^* is the specific ideal gas constant, T the absolute temperature of the gas, p the absolute pressure, ρ_o gas density at the linearisation point, p_o and T_o are the values of absolute pressure and temperature at the same point. Notice the use of the relative pressure $P = p - p_o$, in order to avoid numerical errors due to the large absolute pressure values.
The discrepancy between the ideal and the linearised model is very limited in the typical operating range.
Besides the state equation, the specific energy and enthalpy ones are clearly needed. With a couple of non-relevant assumptions, however reasonably satisfied in this context, such equations are:

$$e = c_vT \tag{4.5}$$

$$h = e + \frac{p}{\rho} = c_pT \tag{4.6}$$

where c_v and c_p are respectively the specific heat capacity at constant volume and pressure, both assumed to be constant.

4.1.3 Thermal exchanges

The thermal power flowing from one volume to another may be computed with the Fourier-like law

$$Q_{1\to 2} = \gamma \frac{A}{d}(T_1 - T_2) \tag{4.7}$$

where γ is the fluid's thermal conductivity, A is the surface shared by the adjacent volumes, d is the distance between the volume centres, and T_x are the temperatures of the volumes.

In addition, when dealing with boundary conditions such as walls, there is a convective heat transfer equation instead of the conductive one. The thermal power flowing can thus be calculated as

$$Q_{\text{wall}\to\text{ vol}} = hA(T_{\text{wall}} - T_{\text{volume}}) \tag{4.8}$$

where h is the convective heat transfer, A is the portion of area shared by the volume and the wall.

4.1.4 Turbolence modelling

For laminar flows, the results provided are natively accurate and reliable. As witnessed by the CFD literature, the same is not true for turbulent flows. The introduction of a *turbulence model* is the most common way to solve this problem, and a variety of these have been studied and implemented.

The main purpose of turbulence models in the building context is to give an approximation of the so called “eddy viscosity”, interpreted as a modification of the fluid's intrinsic dynamic viscosity, computed so as to account for the presence of eddies in the turbulent motion. Those eddies, that may have different sizes, are responsible for the augmented diffusion of fluid particles properties (like velocity). To keep complexity as low as possible, the turbulence model used here is a zero-equation one. This means that no additional partial differential equations are introduced, and only an algebraic relationship is used for computing the eddy viscosity. Such a simplification may jeopardise the precise representation of

velocity fields, but is of less impact on the evaluation of temperature and heat transfer fields, which are the main goal of energy-related simulations.
The viscosity, when dealing with turbulent flows, is thus replaced by the "effective" dynamic viscosity

$$\mu_{\text{eff}} = \mu + \mu_T = \mu + 0.03874\rho\bar{v}l \tag{4.9}$$

that is a sum of the intrinsic fluid dynamic viscosity μ and a turbulent viscosity μ_T, that comes from an algebraic function of local mean velocity $\bar{v}$ (the velocity of the air flowing through the coupling element) and at a length scale l (the distance between the centres of the volumes linked by the coupling element).

4.2 Data centre models

Among the many available configurations for data centre components placement, the chosen one as a reference is that in figure 4.2.

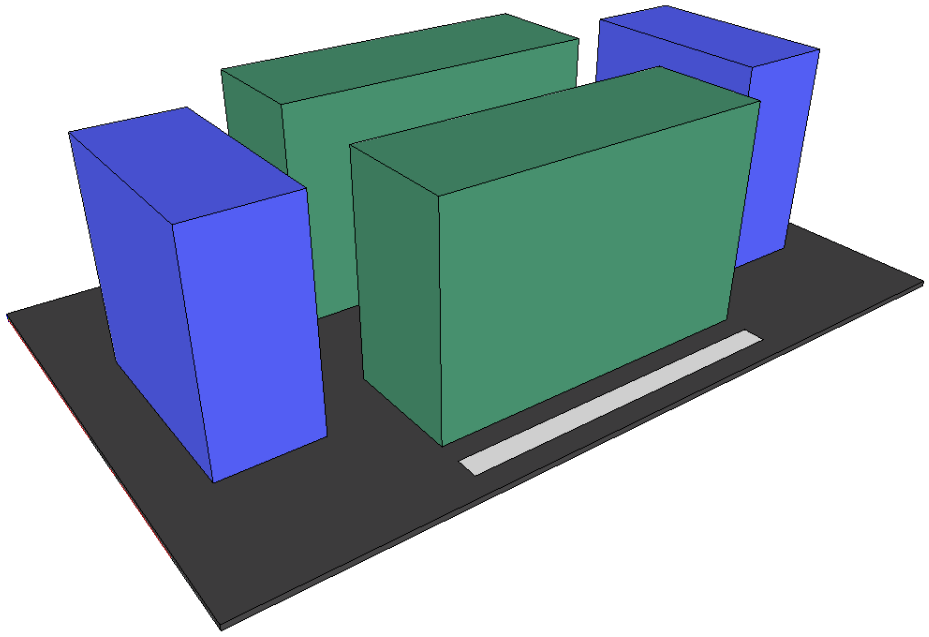
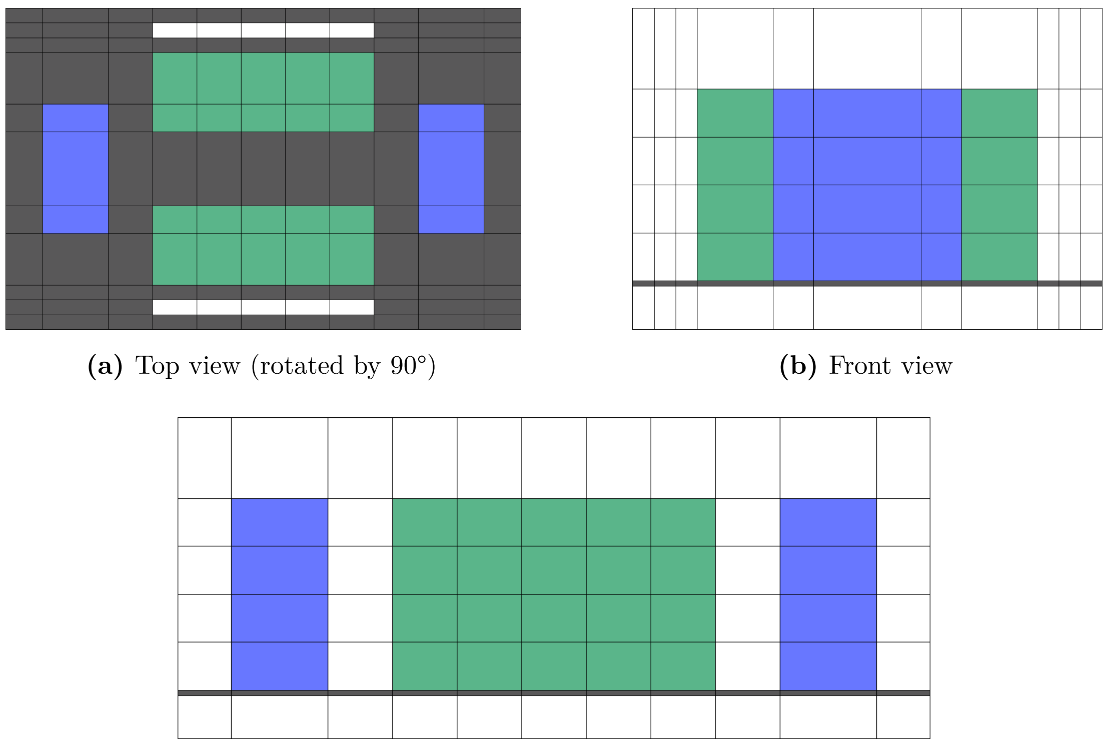


Figure 4.2: Reference data centre room

It features two facing rows made up of five racks, each filled with four blade enclosures. Such a layout induces a hot aisle in the space between the servers,

while the backside of the racks is in a cold aisle.
Additionally, two CRAC units are place transversely to the racks, one per side of the room. In the considered data centre, only one CRAC is supposed to be working, while the other one is ready as a backup upon failure; that means that it doesn't show any difference from a wall. Finally, two rows of open vents in the floor provide the cold air rising from the plenum.
The whole server room is modelled following the design principles depicted in 4.1, thus each component needs to be mapped to the room in terms of volumes and layers.


(a) Top view (rotated by 90°) **(b)** Front view

(c) Side view

Figure 4.3: Data centre mapping to volumes

Figure 4.3 shows the orthographic projections of the grid mapped onto the actual room. To enhance performance, however, a couple of further simplifications can be considered:

- The space between the CRAC units and the wall on their back can be neglected;

- Similarly, the space surrounding the vents (wall-vents and vents-racks) can be ignored;
- The floor can be considered as flat, i.e., setting its thickness to 0 m.
- The space between the CRAC units and the racks can be cancelled.

All of these simplifications don't affect simulation accuracy, except for the last one. Removing that space, in fact, may prevent a the representation of the little air recirculation on the side of the racks.
Furthermore, each element should of course reproduce its own behaviour.

4.2.1 Server

A server is quite a complicated piece of hardware. Commercial products are, in fact, designed to address multiple issues: besides load elaboration, which is of course the mail goal, a server should provide interfaces for storage and network communication, smart power management (e.g., ACPI states), virtualisation and so forth.
For the purpose of this work, however, there's no need to model accurately the hardware part, since a simple model, yet accurate concerning thermal issues, would be enough. Hence, the power consumption of the server is

$$W = W_{\text{idle}} + f \cdot l \cdot W_{\text{busy}} \tag{4.10}$$

where W_{idle} is the power consumed when the server is idle, W_{busy} is the power dynamically consumed according to the frequency f and to the load l.
Both frequency and load are continuous quantities in the interval $[0, 1]$. This assumption, in addition to equation 4.10, implicitly states that the amount of work done by a server is

$$0 \leq l \cdot f \leq 1 \tag{4.11}$$

With reference to a Dell PowerEdge M610 blade server [13], the actual power consumption can be devised by one of SPEC's benchmarks [8]. Blade servers must be enclosed in some kind of chassis, and the chosen one is Dell PowerEdge M1000e [12].
Since servers cannot exist as standalone units along the room, racks should be modelled as well. Rack's model does nothing more than its physical counterpart: it just holds servers and routes control signals coming from the controller. The Rack Enclosure adopted as a reference herein is Dell PowerEdge 4220 [11].

4.2.2 CRAC

In accordance with the project's aim, there's no need to model the complex thermodynamic phenomena involved in a HVAC system. The only task performed by the CRAC's model is to inject an amount of power computed in order to keep its outlet temperature as close as point to the set point.
Given the above, the modular structure of the libraries involved allows a more detailed model to be quickly integrated upon need, e.g., a model taking into account the hydraulic part, too.
The CRAC adopted as a reference here is STULZ CyberAir 2 DX 531 A [10].

4.3 Control strategies

The controller itself is quite a simple object: it's designed to regulate the temperature set point of the CRAC and to modulate the fan velocities of the servers and the CRAC. A buffer stores the number of requested jobs and delivers them to a load balancer, at an elaboration rate decided by the balancer itself. This is the place where the "magic" happens, i.e., jobs are mapped to the needed resources (servers).
Finally, a protection mechanism for the servers must be supplied. Servers, in fact, must work in quite a strict operating range (for those in 4.2.1, said range is 10 °C

to 35 °C). When temperature rises above the maximum allowed $T_{\max}$, the server is completely shut down. Additionally, since a complete shut down to start up cycle is expensive in term of components' strain (especially concerning hard disks), the controller tries to prevent its inlet temperature from surpassing the threshold by scaling the frequency and lowering the load. The jobs lost in such scenario are then pushed back to the queue.
Said scheme is known as a *two-level* hierarchical control architecture. No matter what the load balancer determines, its decision is *overridden* by the upper level controller.

4.3.1 Uniform balancing

This strategy is the most simple (and silly) that can be adopted: every server is left on and the load is equally partitioned across all the servers.
Such a schedule may work well when the server runs constantly a high load level, and when overheating is not an issue to deal with. In such a scenario there's not much left to be optimised. Luckily, real data centres usually behave in a different way.

4.3.2 Minimal balancing

This is some kind of a smarter strategy. The number of servers allocated is obtained as the ceiling[2] of the requested load, then such load is equally partitioned across all the server currently working.
Such a schedule still shows some flaws, the most important of which is that it does not distinguish the servers being allocated according to some metric, i.e., the temperature at inlet. Something even smarter should consider a server rank induced by some measurable metric.

[2]The ceiling function $\lceil x \rceil$ maps a real number to the smallest following integer. More precisely, it is the smallest integer not less than x.

4.3.3 Smart balancing

The smartest balancing policy implemented herein relies on the ordering among the servers induced by the inlet temperatures. Servers are classified as follows:

Those who *can* If inlet temperature is $T_{\text{min}} \leq T < T_{\text{crit}}$, where T_{crit} is a critical threshold above which some countermeasure should be taken to prevent overheating;

Those who *may* If $T_{\text{crit}} \leq T < T_{\text{max}}$. In this case servers can still operate at reduced frequency and load (thus, power consumption), but overheating is close to happen;

Those who *can't* When the server isn't within the operating range, then it must be shut down.

The allocation policy is designed in order to scan the first two classes (those within operating range) and turn on as many servers as needed. This is the same as in 4.3.2, with the small difference that the servers are turned on according to their ranking.
To further improve such strategy, a two-pass scanning may be adequate. The first pass, in fact, will turn on just the servers that were on also in the previous allocation scheme; the second pass will deal with the remaining load, if any. This trick will reduce the number of stop-restart, thus (hopefully) extending the time-to-failure of the machine.
The allocation algorithm is not continuously applied, but it is recalculated every T_s seconds, where T_s is assessed in order to gain a trade off between computation costs and timeliness. The new policy becomes operative after a T_i time frame, representing the time a server needs to be initialised.
On the counterpart, load is instantly redistributed according to the current partition scheme. Higher load simply goes to the coldest server, and so forth.

Chapter 5

Implementation

The models were realised around the aforementioned subzonal models, whose actual implementation won't be shown for brevity. Further details can be found in [20], [5] and [6]. However, section 5.1 will still deal with such models, showing the objects made in order to enhance said framework.
Afterwards, the actual models of data centre's elements and controllers will be introduced.
This chapter isn't meant to describe every single line of code as it would not be meaningful at all, but its main purpose is to roughly describe how the models interact, in order to allow a reader to easily extend any model. Hence, sometimes it may happen that some code snippets will be slightly different from the implemented ones. Additionally, due to the three-dimensional nature being represented, some functions inherently share similar code for each direction: only one will be described.

5.1 Subzonal models

Subzonal models are designed to deal with thermal flows within a room, making it possible, for example, to stuck velocities in some layers (e.g., to place a wall) or to play around with exchange coefficients (e.g., to make the wall insulated). In order to achieve a modular and effective structure, some functions were written to represent the behaviour of obstacles, fans and holes inside the room.

These functions rely on a couple of convenient data structures.

5.1.1 Data structures

The basic structure is the `Box`, and it just represents a set of contiguous volumes.

`record Box`

```
Integer i1(min=1), j1(min=1), k1(min=1);
Integer di(min=0), dj(min=0), dk(min=0);
final Integer i2 = i1 + di - 1;
final Integer j2 = j1 + dj - 1;
final Integer k2 = k1 + dk - 1;
```

Code Snippet 2: Box of volumes

When placing an item somewhere in the room, the easiest approach is to provide the first coordinate (e.g., `i1`) and the number of volumes which the item spans over in that direction (e.g., `di`). Hence, the coordinates of the second point is `i1 + di - 1`. “First” and “second” are named according to the internal ordering given by the subzonal framework itself: coordinates grow from West to East along X axis, from South to North along Y axis, and from Bottom to Top along Z axis. Since using two grids, one for volumes and one for layers, can be sometimes misleading, it comes handy to write the functions to get the surface volumes or layers with reference to a box.

`function GetSurfaceVolumes`

```
if which == Face.W or which == Face.E then
  boxSurface := Box(
    i1 = if which == Face.W then (if not outOfBox then box.i1 else box.i1-1)
      else (if not outOfBox then box.i2 else box.i2+1),
    di = 1,
    j1 = box.j1, dj = box.dj,
    k1 = box.k1, dk = box.dk);
end if;
```

Code Snippet 3: Box whose coordinates represent the requested surface volumes

This function can handle a special request: it can return the volumes immediately out of the box on the desired surface. Such a functionality comes useful when retrieving the inlet temperature of a server.

```
function GetSurfaceLayers

if which == Face.W or which == Face.E then
  boxSurface := Box(
    i1 = if box.di == 0 then box.i2
      else (if which == Face.W then box.i1-1 else box.i2),
    di = 1,
    j1 = box.j1, dj = box.dj,
    k1 = box.k1, dk = box.dk);
end if;
```

Code Snippet 4: Box whose coordinates represent the requested surface layers

A box whose `di` is zero is used in a tricky way to handle dimensionless items (e.g., floor is allowed to have zero thickness, thus `dk == 0`, and there's only one layer along Z direction).

The operation of placing something in the room deals with more than one variable, that's why a convenient structure should be created in order to keep all these things together.

```
record FixedLayers

parameter Integer I(min=2), J(min=2), K(min=2);
Boolean     fixX[I-1,J,K],  fixY[I,J-1,K],  fixZ[I,J,K-1];
SI.Velocity velX[I-1,J,K],  velY[I,J-1,K],  velZ[I,J,K-1];
Boolean     insX[I-1,J,K],  insY[I,J-1,K],  insZ[I,J,K-1];
Boolean    convX[I-1,J,K], convY[I,J-1,K], convZ[I,J,K-1];
```

Code Snippet 5: Information about fixed layers

This structure is made of four fields per direction:

- `fix` is a boolean flag showing whether the velocity of the corresponding layer is stuck at some value or not;
- `vel` is the (constant) value of velocity if `fix == true`; if not it's just neglected;

- `ins` represents an insulated layer: no thermal exchange is allowed across it;
- `conv` tells whether the layer will perform convective heat exchange instead of conductive; if `ins == true` this flag is overridden, thus neglected.

5.1.2 Obstacle

This is one of the three functions designed to fill the fields of a `FixedLayers` variable in order to reproduce the behaviour of an insulated object.
First, the data structures are initialised in a convenient way.

function PlaceObstacle

```
fixX  := fill(false, I-1, J, K);
velX  := fill(0,     I-1, J, K);
insX  := fill(false, I-1, J, K);
convX := fill(false, I-1, J, K);
```

Code Snippet 6: Function to place an obstacle – Initialisation

An additional flag (more precisely, a flag *per axis*) allows to fix the velocity of the inner elements, too. Finally, the layers on each surface of the box must have their velocities stuck to zero and the insulation flag set to `true`.

function PlaceObstacle (cont.)

```
if di > 1 and not emptyX then
  fixX[i1:i2-1, j1:j2, k1:k2] := fill(true, di-1, dj, dk);
  insX[i1:i2-1, j1:j2, k1:k2] := fill(true, di-1, dj, dk);
end if;

for face in {Face.W, Face.E} loop
  s := GetSurfaceLayers(box=box, which=face);
  if s.i1 > 0 and s.i2 < I then
    fixX[s.i1:s.i2, s.j1:s.j2, s.k1:s.k2]  := fill(true, 1, box.dj, box.dk);
    insX[s.i1:s.i2, s.j1:s.j2, s.k1:s.k2]  := fill(true, 1, box.dj, box.dk);
    convX[s.i1:s.i2, s.j1:s.j2, s.k1:s.k2] := fill(true, 1, box.dj, box.dk);
  end if;
end for;
```

Code Snippet 6: Function to place an obstacle – Layers

5.1.3 Fan

A fan is simply modelled impressing a fixed velocity to the inlet and outlet faces of a box. Initialisation phase shares the same philosophy as the `PlaceObstacle` function just described.

`function PlaceFan`

```
fixX  := fill(false, I-1, J, K);
velX  := fill(0,     I-1, J, K);
insX  := fill(true,  I-1, J, K);
convX := fill(false, I-1, J, K);
```

Code Snippet 7: Function to place a fan – Initialisation

Fan's velocity is given as an absolute value, and the in/out sides are explicitly provided. The actual placement must compute the sign of `FixedLayers.vel` according to the direction of the flow. No action should be taken if fan velocity is zero.

`function PlaceFan (cont.)`

```
for side in {inlet, outlet} loop
  if side > Face.NO then
    sign := if side == inlet then +1 else -1;
  else
    sign := if side == outlet then +1 else -1;
  end if;
  s := GetSurfaceLayers(box=box, which=side);
  if (side == Face.W and i1 > 0) or (Face.E and i2 < I) then
    fixX[s.i1:s.i2, s.j1:s.j2, s.k1:s.k2]  := fill(true, 1, dj, dk);
    velX[s.i1:s.i2, s.j1:s.j2, s.k1:s.k2]  := sign * fill(fanSpeed, 1, dj, dk);
    insX[s.i1:s.i2, s.j1:s.j2, s.k1:s.k2]  := fill(false, 1, dj, dk);
    convX[s.i1:s.i2, s.j1:s.j2, s.k1:s.k2] := fill(true, 1, dj, dk);
  end if;
end for;
```

Code Snippet 7: Function to place a fan – Surface layers

As in `PlaceObstacle`, one may want to fix the velocity of internal layers, too. This task can be accomplished when inlet and outlet lay on the same direction.

The main reason for doing this is to help the solver by removing a variable from being computed.

```
function PlaceFan (cont.)
if inlet + outlet == Face.NO then
  sign := if inlet > Face.NO then +1 else -1;
  if (inlet == Face.W or inlet == Face.E) and di > 1 then
    fixX[i1:i2-1, j1:j2, k1:k2] := fill(true, di-1, dj, dk);
    velX[i1:i2-1, j1:j2, k1:k2] := sign * fill(fanSpeed, di-1, dj, dk);
    insX[i1:i2-1, j1:j2, k1:k2] := fill(false, di-1, dj, dk);
  end if;
end if;
```

Code Snippet 7: Function to place a fan – Inner layers

5.1.4 Hole

Structures' initialisation is nothing new.

```
function PlaceHole
fixX  := fill(true, I-1, J, K);
velX  := fill(0,    I-1, J, K);
insX  := fill(true, I-1, J, K);
convX := fill(true, I-1, J, K);
```

Code Snippet 8: Function to place a hole – Initialisation

Code blocks are separated like in other functions. First layers to deal with are the inner ones.

```
function PlaceHole (cont.)
if di > 1 then
  fixX[i1:i2-1, j1:j2, k1:k2]  := fill(false, di-1, dj, dk);
  insX[i1:i2-1, j1:j2, k1:k2]  := fill(false, di-1, dj, dk);
  convX[i1:i2-1, j1:j2, k1:k2] := fill(false, di-1, dj, dk);
end if;
```

Code Snippet 8: Function to place a hole – Inner layers

Finally, surface layers have to be taken into consideration.

function PlaceHole (cont.)

```
for hole in holes loop
  s := GetSurfaceLayers(box=box, which=hole);
  if (hole == Face.W and i1 > 0) or (hole == Face.E and i2 < I) then
    fixX[s.i1:s.i2, s.j1:s.j2, s.k1:s.k2]  := fill(false, 1, dj, dk);
    insX[s.i1:s.i2, s.j1:s.j2, s.k1:s.k2]  := fill(false, 1, dj, dk);
    convX[s.i1:s.i2, s.j1:s.j2, s.k1:s.k2] := fill(false, 1, dj, dk);
  end if;
end for;
```

Code Snippet 8: Function to place a hole – Surface layers

5.1.5 Join obstacles, fans and holes

The functions just written are not enough to completely represent a data centre by themselves. Conversely, providing a way to combine their actions could provide a wider range of serviceable models.

In fact, for example, a simple server can be modelled as an obstacle with a fan, or a floor can be thought as an obstacle with adequate holes for vents.

function JoinTogetherOHF

```
joined.fixX  := (obstacle.fixX or fan.fixX) and hole.fixX;
joined.velX  := fan.velX;
joined.insX  := obstacle.insX and fan.insX and hole.insX;
joined.convX := (obstacle.convX or fan.convX) and hole.convX;
```

Code Snippet 9: Join obstacles, holes and fans

5.2 Data centre

To achieve a finer parametrisation of the computer room, some convenient structures can be used to easily interchange one blade with another, rather than a CRAC with another one.

Since the assumption made is here is that a `Rack` is filled with blade servers, the structures needed must represent the `Enclosure` and the `Blade` itself. In order not to add unwanted complexity to the models, the blade enclosure is supposed not to draw any power, which is entirely drawn by the blades. An enclosure can be filled somewhere between empty and full. On the counter, no fan is bound to the blades – which is quite realistic – and the whole airflow comes from the fans shipped with the blade enclosure. These fans can be swapped, too, that's the reason why a simple and convenient structure for the `Fan` is used.

```
record Fan
SI.VolumeFlowRate CMS;
```

Code Snippet 10: Fan

```
record Blade
SI.Power W_idle, W_busy;
SI.Temp_K T_min, T_max, T_crit;
```

Code Snippet 11: Blade server

```
record Enclosure
Integer blades, fans;
SI.Distance height;
```

Code Snippet 12: Blade enclosure

```
record Rack
SI.Distance width, height, depth;
```

Code Snippet 13: Rack chassis

Additionally – besides width, height, depth and fan speed – the CRAC model needs to know its COP (possibly obtained by converting from EER).
Since efficiency is variable with input/output temperatures, COP is implemented as a table where first input is the temperature at inlet, while second input is that at the outlet. COP is smoothly interpolated according to the actual input signals.

```
record CRAC
SI.Distance width, height, depth;
SI.VolumeFlowRate fanVol;
final SI.Velocity fanVel = fanVol / (width*depth);
Real tableCOP[:,:];
```

Code Snippet 14: Computer Room Air Conditioning

Finally, physical dimensions are stored into the `DataCentre` record.

```
record DataCentre
parameter Integer nRows, nRack, nServer, nCrac, nNoz;
SI.Distance floorSpace, floorHeight, nozzleBase;
SI.Distance xSide, xNozzleRack, xRackCrac;
SI.Distance ySide, yCracRack;
SI.Distance zTop;
Face serverIn[nRows], serverOut[nRows], cracIn[nCrac], cracOut[nCrac];
```

Code Snippet 15: Data centre room

5.2.1 Floor

This is the simplest model among those making up the data centre. Floor doesn't exchange heat: it prevents heat exchange, instead. Its only task is to place an obstacle and open the holes for vents and the CRAC.

```
model Floor
Box cracsBase[nCrac] = DC.Records.Box(
  cracs.i1, cracs.j1, floor.k1, cracs.di, cracs.dj, floor.dk);
FixedLayers obstacle = PlaceObstacle(
  I, J, K, floor, emptyX=false, emptyY=false, emptyZ=false);
FixedLayers holes[nNoz+nCrac] = PlaceHole(
  I, J, K, cat(1, nozzles, cracsBase), {Face.BO, Face.TO});
FixedLayers hole = JoinTogetherH(
  I, J, K, nNoz+nCrac, holes);
```

Code Snippet 16: Floor model

5.2.2 Server

The model of server covers both the roles of the blade server and enclosure. Unlike the other models, this is one whose graphical part is representative of its actual implementation. Server model is pictured in figure 5.1.

Power model is that shown in 4.2.1: actual consumption is obtained by multiplying current frequency and load. The server is also designed so that an input signal can control the fan speed, just the way it usually happens.

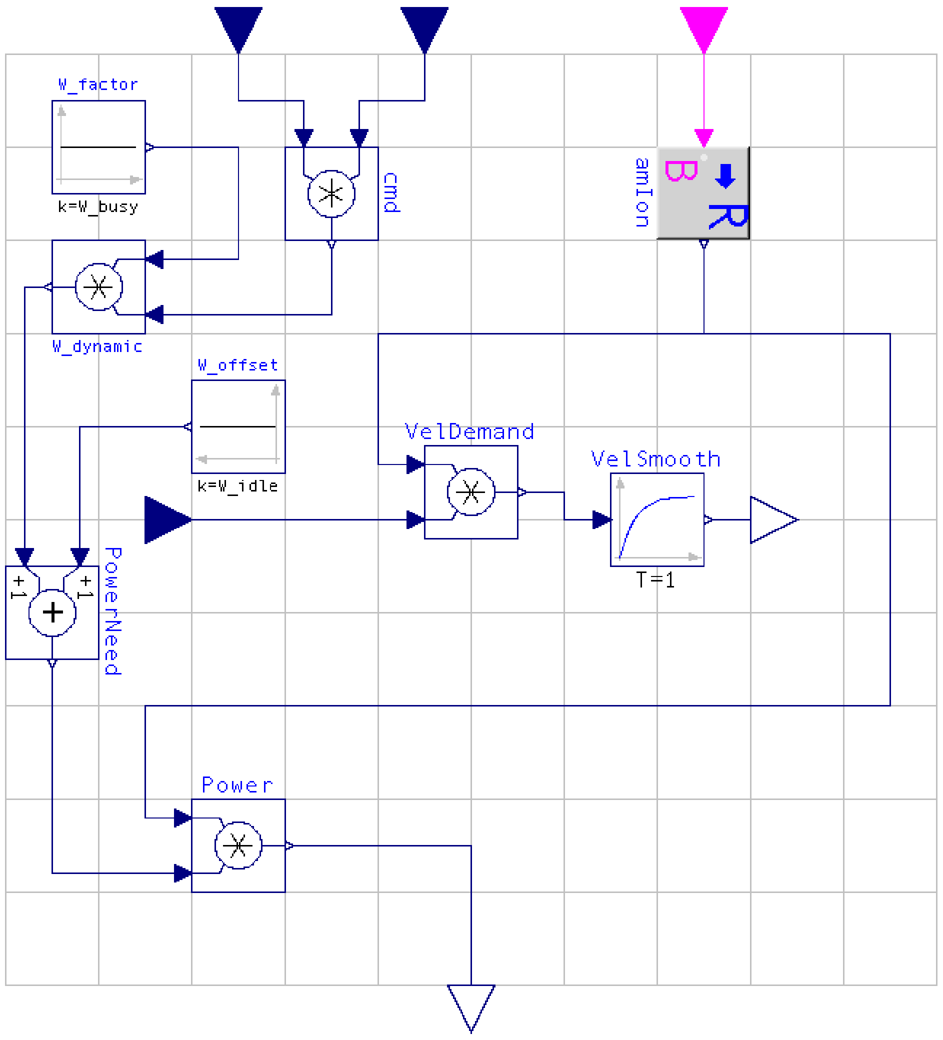


Figure 5.1: Server model

Another input signal determines whether the server is turned on or not. In the latter case, no power consumption takes place and the fan is not working.

This model doesn't rely on any feature of subzonal models, so that it could easily be swapped with another one, hopefully more sophisticated.

5.2.3 Rack

As stated in 4.2.1, rack's model holds servers and routes control signals. This comes when clean looking at figure 5.2.

The signals coming from the servers are routed to the room through the usage of

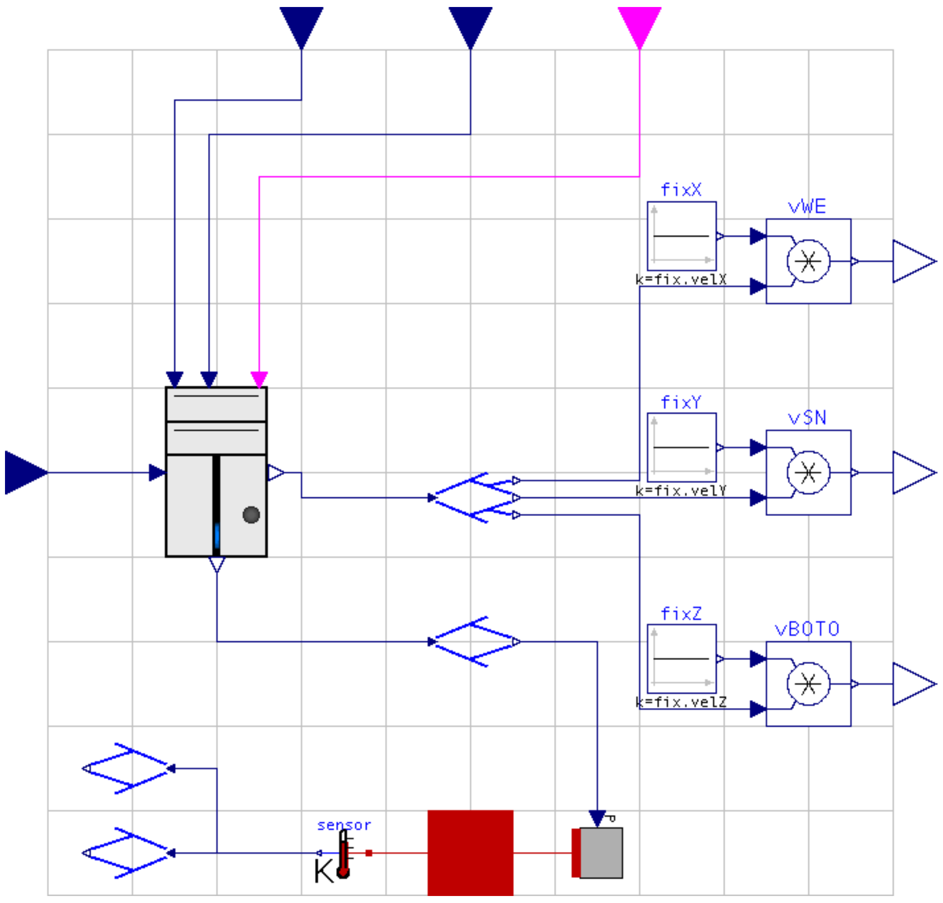


Figure 5.2: Rack model

some adapters: one first kind maps each server to a volume (or more) in the room, another one performs the mapping from the server to the corresponding layers in the room. The opposite operation (from room to servers) can be performed as well.

The mapping between racks and room is quite straightforward. The algorithm scans the box enclosing the racks' row, subsequently computing current rack and server, then its value is assigned to the corresponding volume.

```
function RackToRoom
roomSide := fill(0, I, J, K);
b := fill(0, nRows, nRack, nServer);
for i in i1:i2 loop
  for j in j1:j2 loop
    rack := 1 + div(j-j1, div(dj, nRack));
    for k in k1:k2 loop
      server := 1 + div(k-k1, div(dk, nServer));
      b[row, rack, server] := b[row, rack, server] + 1;
      roomSide[i,j,k] := rackSide[row, rack, server, b[row, rack, server]];
    end for;
  end for;
end for;
```

Code Snippet 17: Racks to room adapter

The same as above happens for layers' mapping. The same idea is repeated for each direction, mapping the rack to every layer belonging to the box.

```
function RacksToLayers
roomSideX := fill(0, I-1, J, K);
for i in i1-1:i2 loop
  for j in j1:j2 loop
    rack := 1 + div(j-j1, div(dj, nRack));
    for k in k1:k2 loop
      server := 1 + div(k-k1, div(dk, nServer));
      if i > 0 and i < I then
        roomSideX[i,j,k] := rackSide[row, rack, server];
      end if;
    end for;
  end for;
end for;
```

Code Snippet 18: Racks to layers adapter – X layers

When mapping layers along Y axis, the rack number calculation must be adapted in order not to lose one layer.

function RacksToLayers (cont.)

```
rack := max(1 + div(j-box.j1, div(box.dj, nRack)), 1);
```

Code Snippet 18: Racks to layers adapter – Y layers (rack number)

Same thing happens for the server in Z-axis mapping.

function RacksToLayers (cont.)

```
server := max(1 + div(k-box.k1, div(box.dk, nServer)), 1);
```

Code Snippet 18: Racks to layers adapter – Z layers (server number)

5.2.4 CRAC

The model of the CRAC is conceived to set the temperature of some volumes to the desired value. More precisely, since reference CRAC's airflow is down-rising, the temperature of the lowermost row of volumes is fixed to the desired set point. Since subzonal models behave awkwardly when blocking temperatures to a fixed value, this model makes use of a smart trick: a standard PI controller maintains the desired temperature in each volume by injecting the correct amount of power. The same adapters as in 5.2.3 were written. No new concept is introduced, then their implementation will be skipped.

5.2.5 DataCentre

The models just introduced are combined with subzonal ones in order to assemble the actual data centre model, as shown in figure 5.4.
The parameters of a Room model must be populated with information coming from `FixedLayers` variables, then those stored inside the room's elements must be combined using the preposed function.

function JoinTogetherRCF

```
joined.fixX  := rack.fixX or crac.fixX or floor.fixX;
joined.velX  := rack.velX + crac.velX;
joined.insX  := rack.insX or crac.insX or floor.insX;
joined.convX := rack.convX or crac.convX or floor.convX;
```

Code Snippet 19: Join Floor, Rack and CRAC

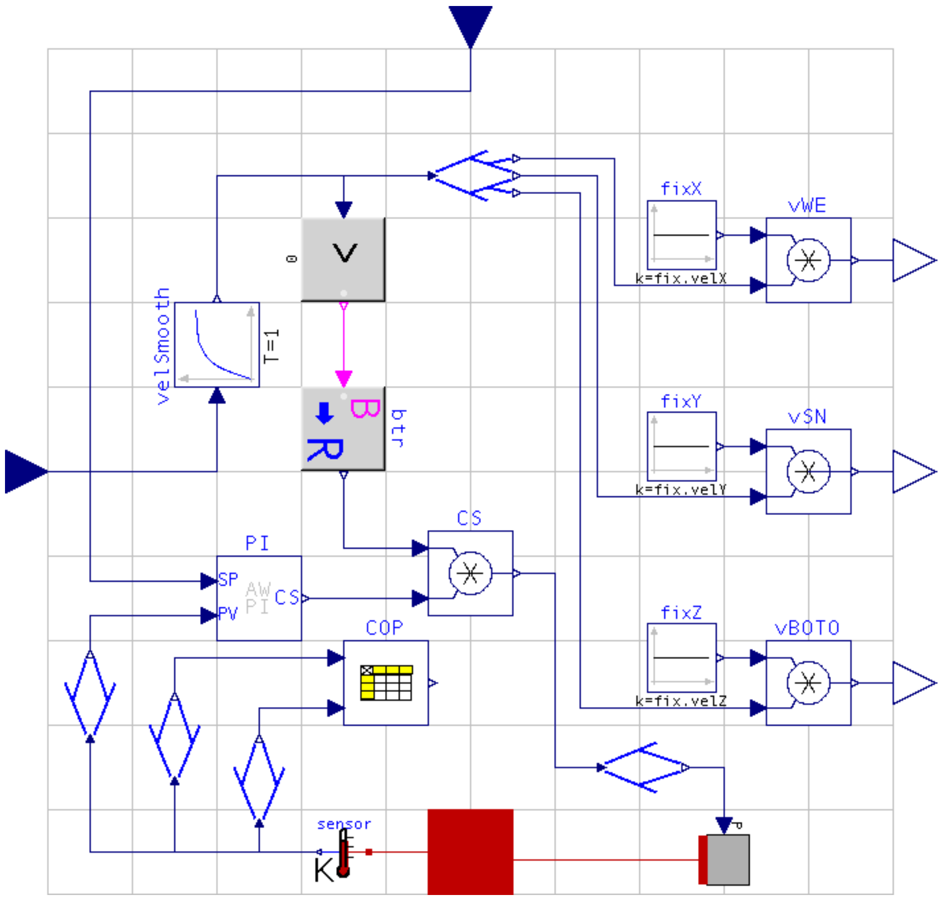


Figure 5.3: CRAC model

The room is surrounded by adiabatic wall, thus not exchanging, and it's connected to the heat terminals of the rack and CRAC's models. Their actual velocities are also added up and connected to the room to induce a velocity field.

5.3 Controller

The control architecture is that described in 4.3, and it's shown in figure 5.5. The queue which it relies on is readily implemented.

```
model Queue
der(queue) = push - pop;
pop = min(queue, req);
```

Code Snippet 20: Queue

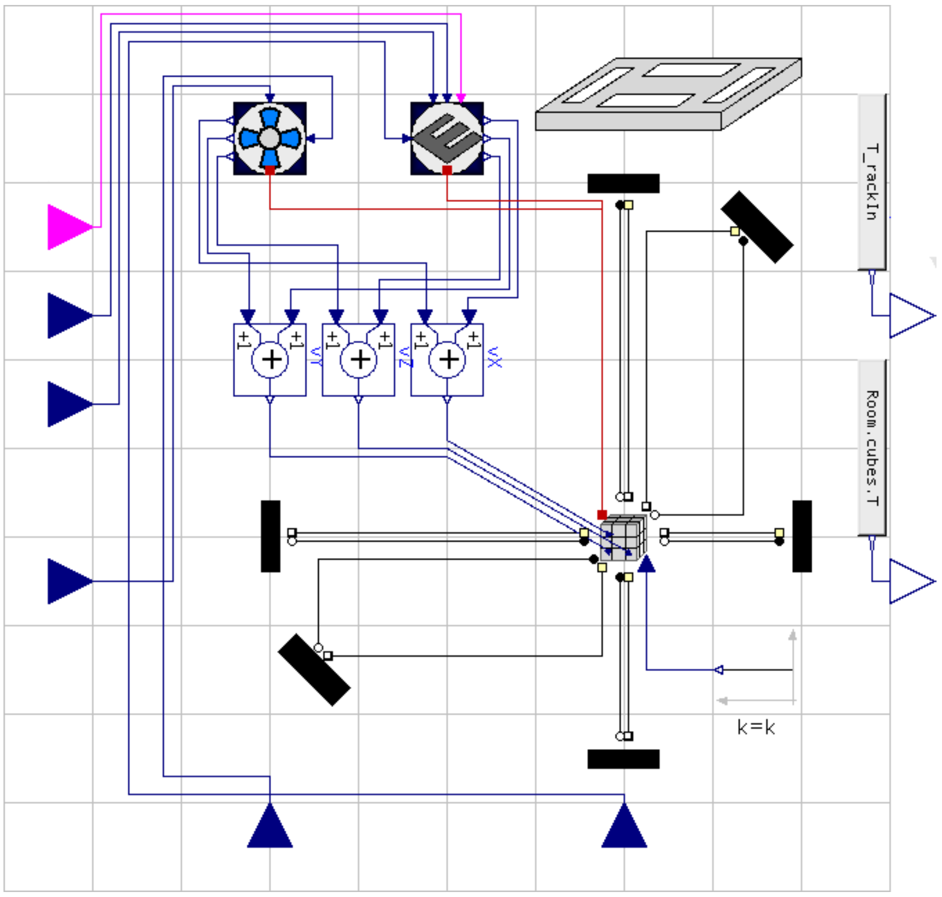


Figure 5.4: Data centre model

The load balancer is the object that decides the amount of load being dequeued and delivered to the right servers. When overheating is detected, the controller can override the balancer's decision by forcing a server shutdown. The amount of rejected load is then pushed back to the queue, and added to the load coming from the outside. Frequency and load are controlled by the means of two simple P controller, while a hysteresis is connected to the on signal. The former solution allows the override action to take place quickly (and efficiently when simulating), while the latter prevents the server to be turned on back again when its inlet temperature is still near the maximum acceptable value.

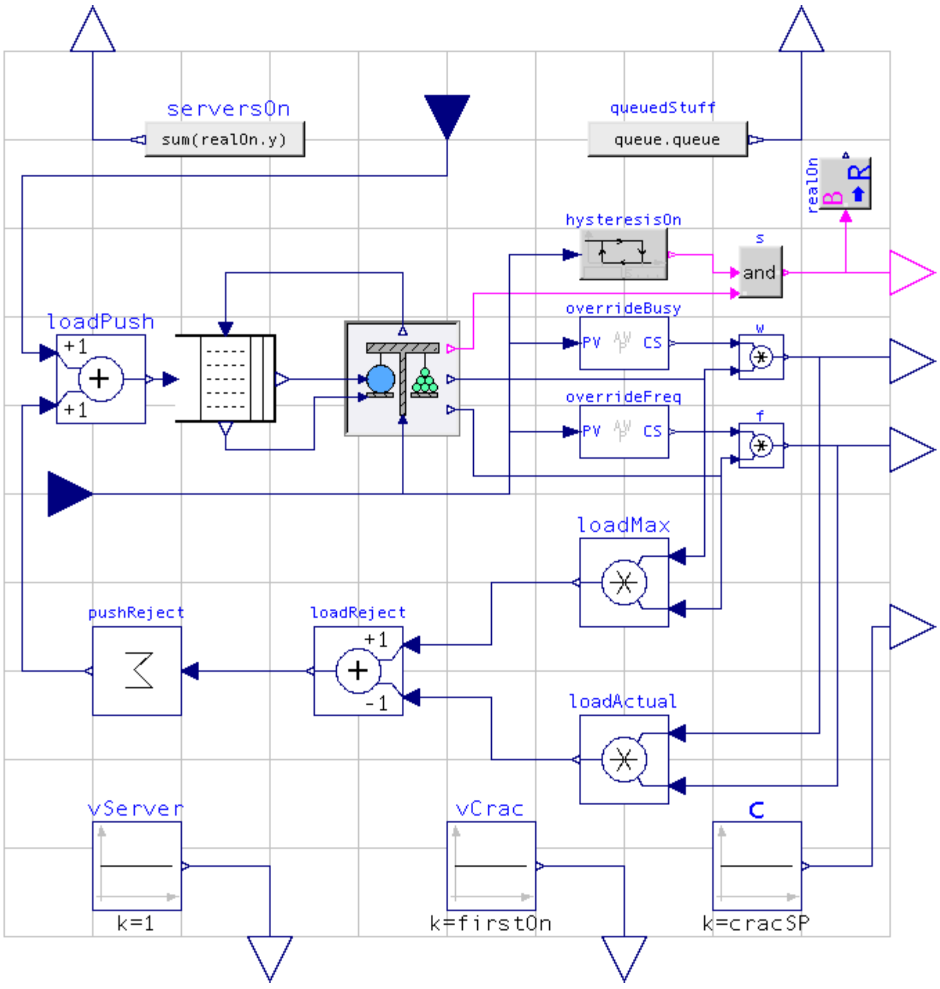


Figure 5.5: Controller model

5.3.1 Load balancer

Load balancer is the component used to do the “dirty” job of allocation and partitioning. Whether a balancer is well designed or not is crucial for the effectiveness of one control strategy rather than another one. In other words, different balancing policies can achieve significant improvements in the efficiency of the entire data centre.

The actual implementation of the strategies discussed in 4.3.1 and 4.3.2 are quite straightforward and won’t be shown, for the sake of brevity.

Smart

This is a balance strategy that tries to resemble that actually being used in data centres. Every T_s (sample time) a new server allocation is computed.

model SmartBalancer

```
when sample(0, sampleTime) then
  new_roles := ClassifyServer(T, Tmin, Tcrit, Tmax);
  tunedTarget := max(
    if keepAtLeastOneOn then 1/(nRows*nRack*nServer) else 0,
    min(1, queueAvg/(nRows*nRack*nServer)));
  new_On := SmartBalance(T, new_roles, old_On
    if not targetAutoTune then target else tunedTarget);
  now_On := old_On or new_On;
end when;
```

Code Snippet 21: Smart balancer - Allocation recalculation

Old and new allocations keep overlapping for a time interval $T_s + T_i$ long, where T_i is the time taken by a server to perform initialisation. No load can be given to any booting server: new allocation will become effective after initialisation time.

model SmartBalancer (cont.)

```
when sample(serverInitTime, sampleTime) then
  now_On := new_On;
  old_On := new_On;
  old_roles := new_roles;
end when;
```

Code Snippet 21: Smart balancer - Allocation application

On the counter, load is continuously given to the servers on at the moment. The partitioning procedure gives higher priority to colder servers, assigning one load unit for each server except for the last one. If current load can be handled by less servers than those currently on, the exceeding ones will be idle.

function SmartLoad

```
busy       := fill(0, size(on,1), size(on,2), size(on,3));
actualLoad := min(load, sum({if on[i,j,k] then 1 else 0 for k, j, i}));
r := 1;
while r <= size(on,1)*size(on,2)*size(on,3) and actualLoad > 0 loop
  i := IndexOf(ranks, r);
  if on[i[1], i[2], i[3]] then
    busy[i[1], i[2], i[3]] := min(1, actualLoad);
    actualLoad := actualLoad - 1;
  end if;
  r := r + 1;
end while;
```

Code Snippet 22: Load redistribution

The allocation algorithm is the same as discussed in 4.3.3.

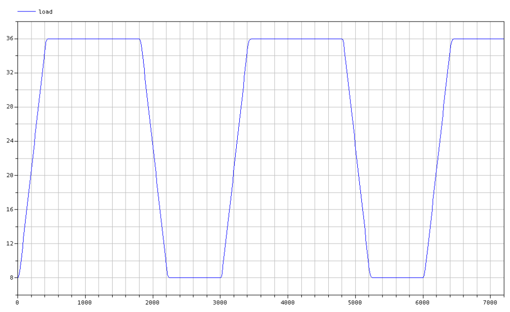
function SmartBalance

```
on         := fill(false, size(T,1), size(T,2), size(T,3));
loadTarget := targetPercentage * size(T,1)*size(T,2)*size(T,3);
r := 1;
while r <= size(T,1)*size(T,2)*size(T,3) and loadTarget > 0 loop
  i := IndexOf(ranks, r);
  if roles[i[1], i[2], i[3]] <> Role.Cant and not on[i[1], i[2], i[3]]
     and (pre_on[i[1], i[2], i[3]] or n <> 1) then
    on[i[1], i[2], i[3]] := true;
    loadTarget := loadTarget - 1;
  end if;
  r := r + 1;
end while;
```

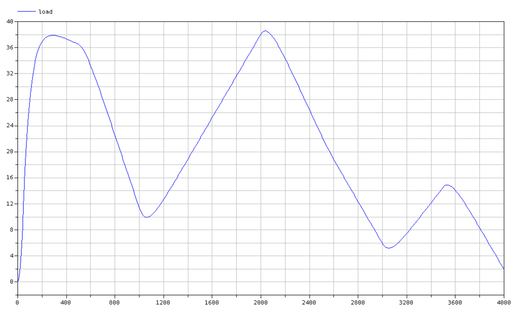
Code Snippet 23: Smart balancing algorithm

5.4 Load

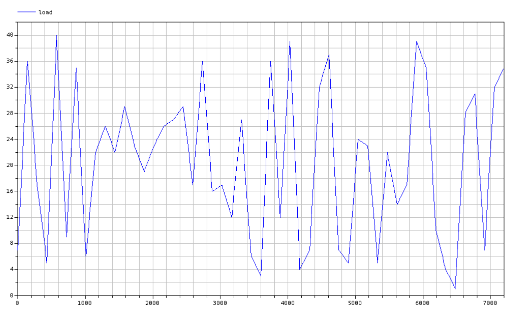
Apart from the easily-understandable full- and no-load models, other available loads are those in figure 5.6.



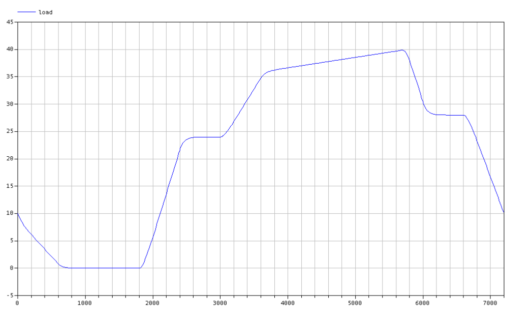
(a) Trapezoidal load



(b) Peaky load



(c) Random load



(d) A possible 24-hour load

Figure 5.6: Available loads

5.5 Validation

Since at the moment of writing there was no availability of physical measures, it was not possible to perform an empirical validation.

However, Bonvini and Leva already made an extensive validation process of their models against real case studies. This premise allows to believe that the data centre's models, which are based on the above mentioned subzonal framework, are correct as long as the structures in 5.1.1 are properly filled.

An intensive analysis of these structures did not reveal any design problem, then the models should be considered as correct.

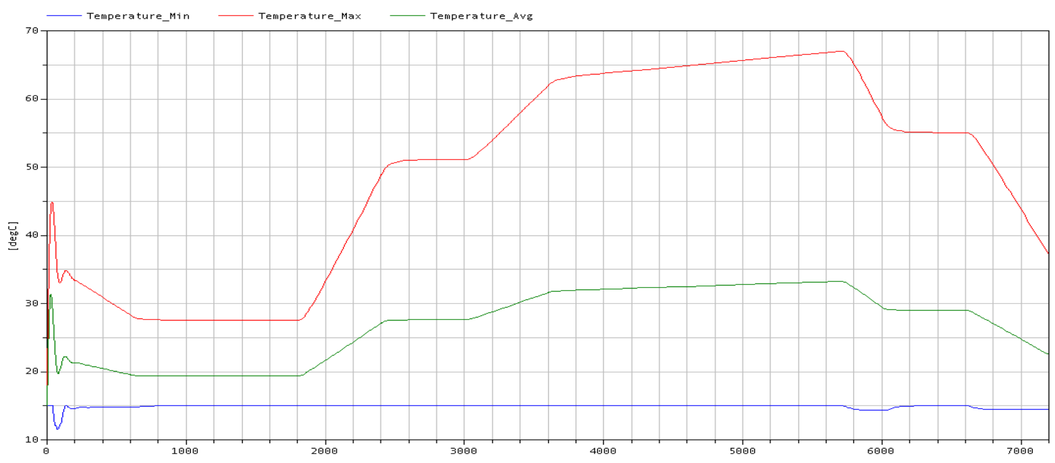
Chapter 6

Case studies

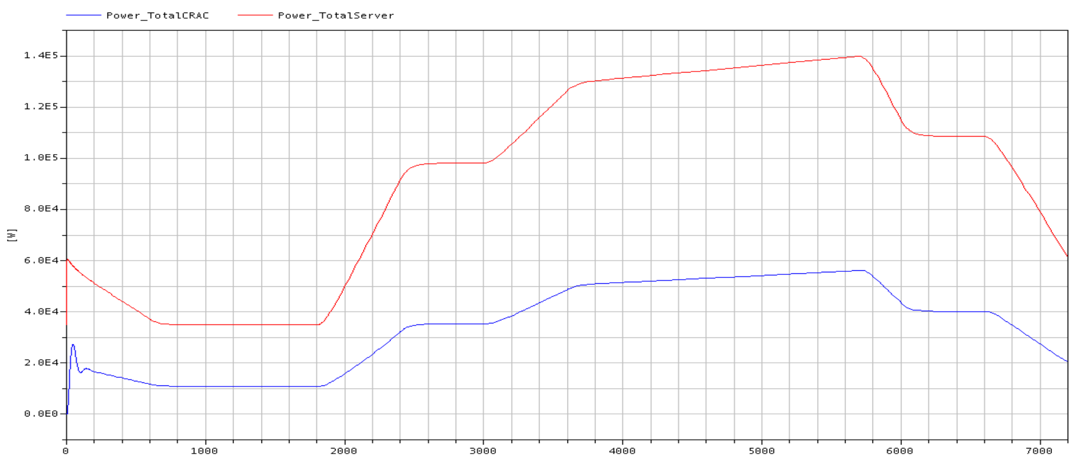
Three representative case studies were designed to test what described so far. In each case, the workload conditions are the same as in figure 5.6d.

6.1 Open loop with uniform balancing

This case is the simplest: all the servers are left on and the total workload is uniformly distributed across all the machines. However, the controller won't take any countermeasure when overheating is detected, allowing the blades to work outside of their operating range.
Surely this is an unrealistic situation, but it was simulated to have a lower bound to which to refer the worst energy efficiency case.
As it comes clear looking at figure 6.1, temperatures across the room are quite high. However, the average keeps quite low on light workload. Many server are accounted for working outside their operating range, but temperatures run back to normal values quite fast when the servers don't work at their maximum power. This is the reason why the mean value of temperatures somehow traces the workload profile.
For analogous reasons, power consumption is the maximum attainable for this data centre configuration.
The only advantage of this allocation scheme is that the buffer is always empty



(a) Temperatures (min, max, avg)



(b) Power consumption (servers, CRACs)

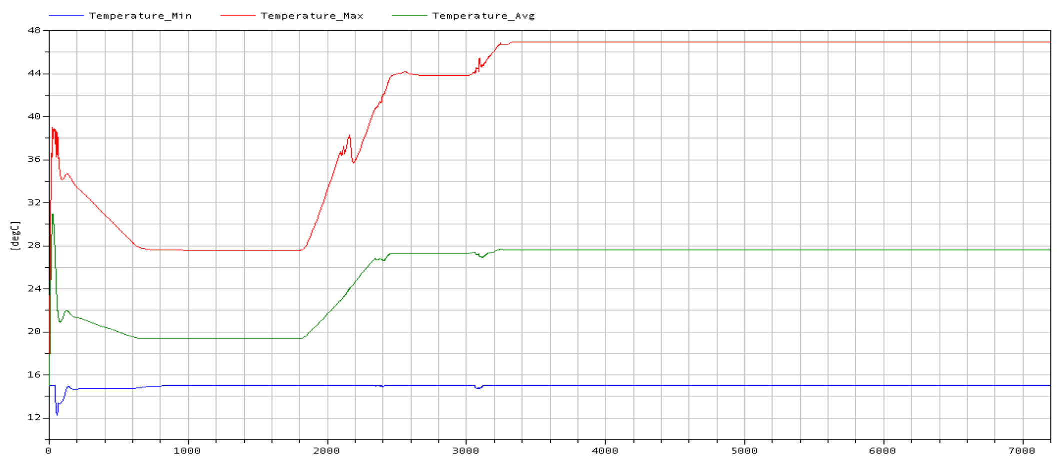
Figure 6.1: Plot for open loop with uniform balancing case

as shown in figure 6.4b, since every request is immediately served regardless of overheating or energy issues.

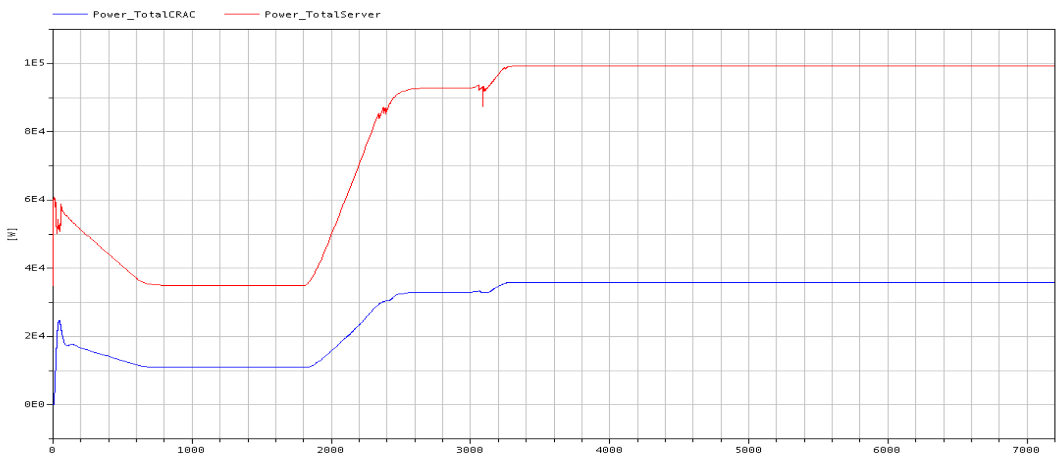
6.2 Closed loop with uniform balancing

The controller used herein is the same as in the previous case, with the exception that it will take a proper action – according to what discussed in 4.3 – upon overheating detection.

Figure 6.2 immediately points out drastic temperature and consumption decrease.



(a) Temperatures (min, max, avg)



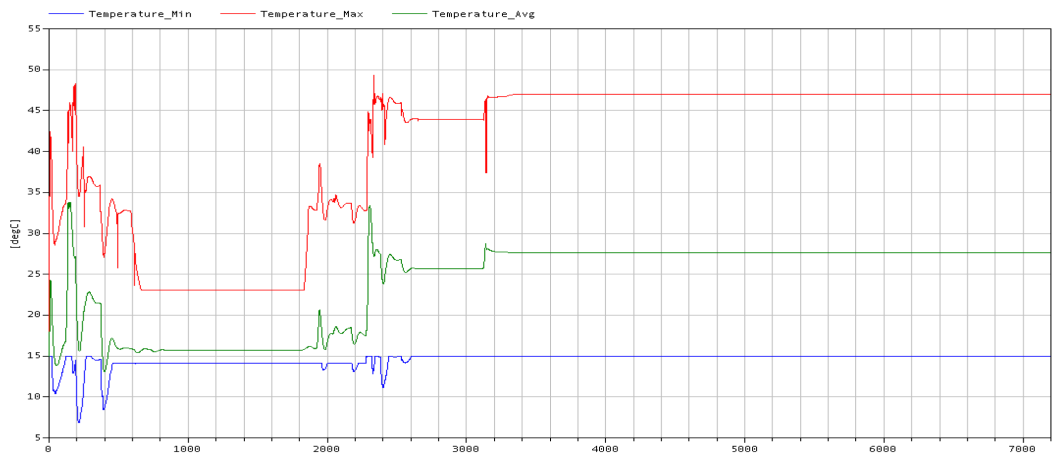
(b) Power consumption (servers, CRACs)

Figure 6.2: Plot closed loop with uniform balancing case

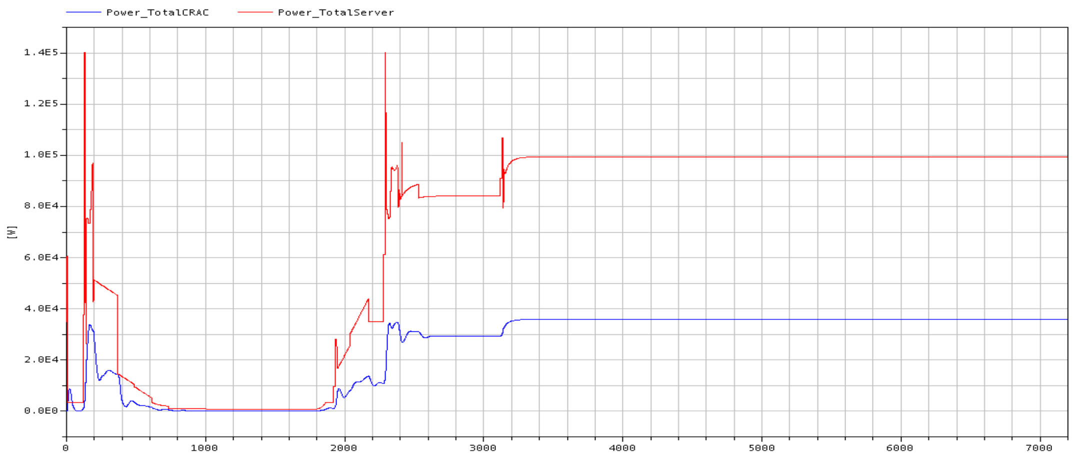
Despite this being desirable from an energetic perspective, servers need to delay some jobs due to overheating protection mechanism.
As can be intuitively understood, this strategy delivers the most load possible with current working conditions, but still many servers are left idle when the data centre is underutilised, consuming more energy than needed.

6.3 Smart balancing

The strategy adopted in this case is that described in 4.3.3.



(a) Temperatures (min, max, avg)



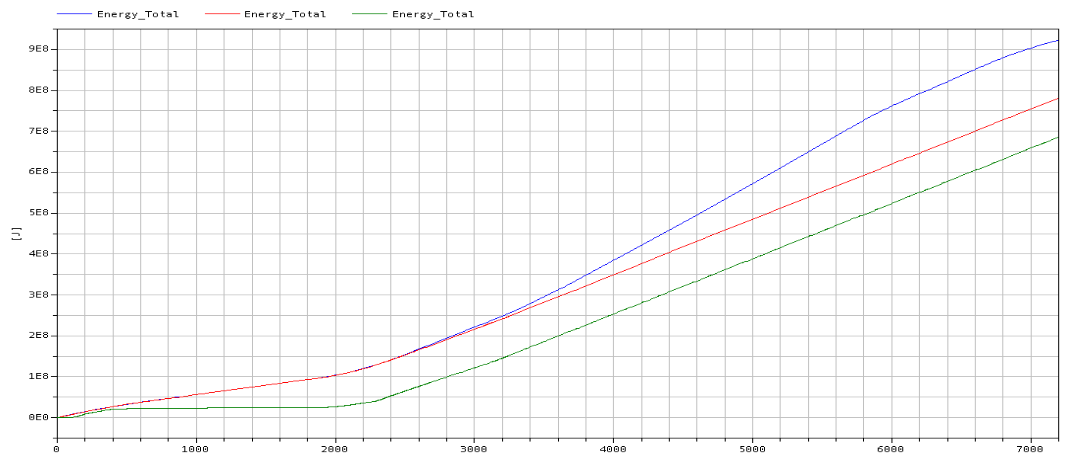
(b) Power consumption (servers, CRACs)

Figure 6.3: Plot smart balancing case

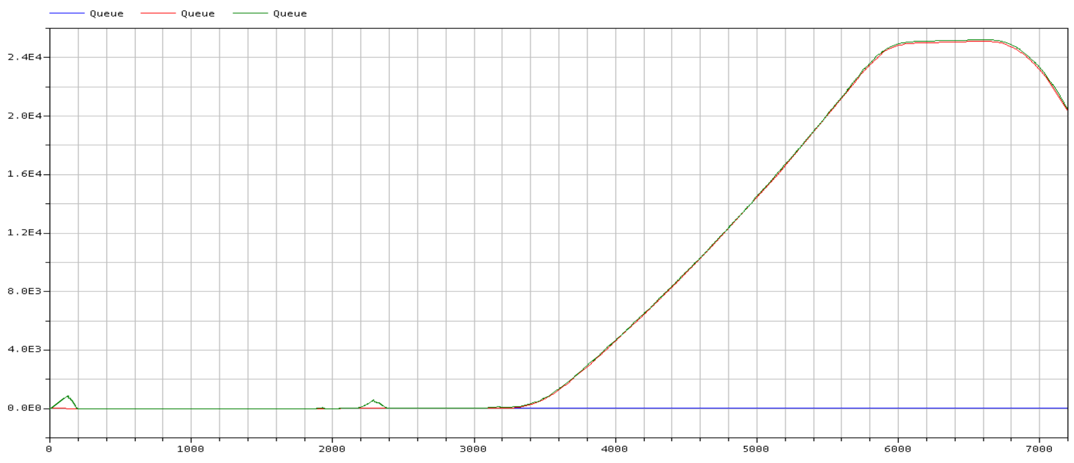
Temperatures result in lower values with reference to previous situations, but power consumption doesn't look pretty much the same, especially with reference to underutilisation zones. In fact, power consumption is the same as in 6.2 when the load is heavy (all the available resources are allocated), whereas low-load regions point out a consistently lower energy waste.

6.4 Comparison

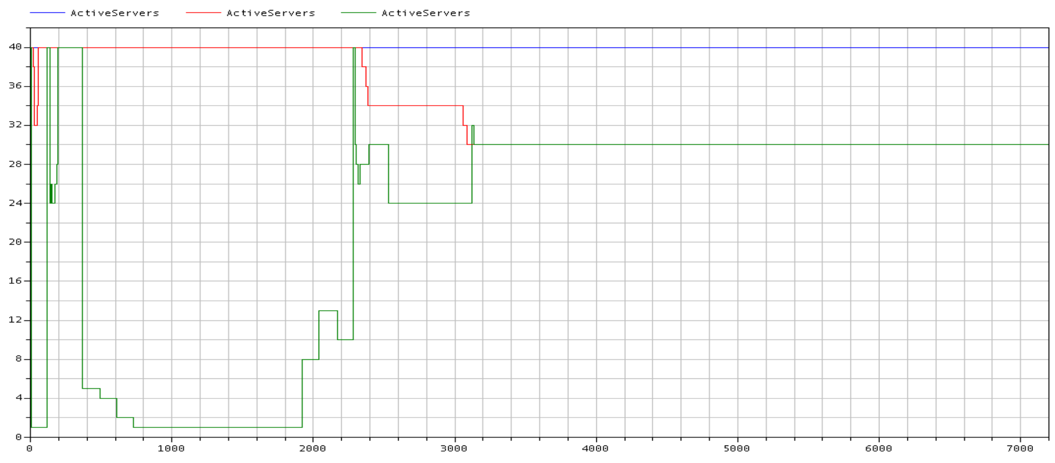
As discussed in previous sections, the case in 6.1 no doubt behaves the worst way. Figure 6.4a highlights said gap in terms of energy consumption. Both the uniform strategies behave similarly in the first part, but they move away as temperatures start to grow and the overheating protection is enforced. Smart strategy (green one) always shows an overall reduced consumption, especially in the aforementioned underutilisation zone.
Despite showing a better energetic behaviour, the smarter strategy manages to delivery almost the same number of jobs as the closed loop uniform one, thus resulting to be the best strategy so far simulated. Open loop uniform strategy (blue one) trivially always keeps the queue empty.
Figure 6.4b also shows a couple of green peaks, whose meaning is inherent to the smart strategy. These peaks, in fact, are symptomatic of a load increase which currently active servers can't react to. When the correct number of servers is subsequently powered on, the peak disappears.
Similarly, as seen in figure 6.4c, the smart strategy always manages to keep a number of servers that is less or the same to what the uniform does, still delivering the same load.



(a) Energy consumption



(b) Queue size



(c) Servers active

Figure 6.4: Case studies comparison

Chapter 7

Conclusion

Although no valid experimental data were available at the moment of writing, temperatures along the room look like their profiles are reasonably coherent with what expected.
Additionally, the outcomes of the three case studies addressed the more sophisticated balance strategy as the most performing one, once again in accordance with the expected results.

7.1 Open issues and further works

As previously mentioned, the lack of any form of experimental validation is the main open issue related to this work.
Despite the proposed methodology works flawlessly, validating the results against measurements would allow it to make a further step towards CFD-like accuracy. Provided that the adopted approach is inherently unable to guarantee the correctness of the velocity field – this not meaning it's always wrong – a reliable framework may open new scenarios, e.g., the search for hot spots. Current library may correctly identify hot spots and airflow design flaws, but there's no real proof of correctness.
A validated library can be used as a supporting tool to improve the methodology developed in [9] and [16]. Indeed CFD simulations could be replaced by more

efficient ones, making it possible to tune the linear model in a significantly shorter time, thus allowing said linear model to be used for online control, e.g., in a Model Predictve flavour.

The data centre and controllers so far implemented are not much more than a proof of concept. They resemble actual data centre, of course, but real ones are typically bigger and more complex (i.e., more than one hot/cold aisle, many rows of rack and so on).

Additionally, the controller may improve energy efficiency also by acting on the set point of the CRAC units and eventually on their outgoing airflow; similarly they can play with servers' fans.

The whole library development, from code to simulations, was carried out using Dymola 6.0b for Linux as a supporting tool. This version dates back to 2006.

Although simulation efficiency is actually quite good, especially when compared to any CFD tool, newer versions of Dymola provide functionalities that should further reduce simulation time. For example, Dymola 7.3 can identify which elements of the Jacobian matrix are identically null, thus avoiding their computation. This functionality proves to be very useful when the model is strongly decoupled, as in the case of this work. Such an improvement is accounted for a 3x factor of simulation time reduction.

A new version of Dymola may also allow a better-looking implementation of some features. Older versions are buggy when dealing with records and some pieces of code are sometimes redundant, or messy, or simply awfully tricky as a result of a workaround for some unwanted issue.

Bibliography

[1] Victor Banuelos. *Managing data centre heat issues.*

[2] Cullen E. Bash, Chandrakant D. Patel, and Ratnesh K. Sharma. Dynamic thermal management of air cooled data centers. In *10th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronics Systems (iTherm)*, May 2006.

[3] Ricardo Bianchini and Ram Rajamony. Power and energy management for server systems. *IEEE Computer*, 37, 2004.

[4] Michele Blazek, Huimin Chong, Woonsien Loh, and Jonathan G. Koomey. Data centers revisited: Assessment of the energy impact of retrofits and technology trends in a high-density computing facility. *Journal of Infrastructure Systems*, September 2004.

[5] Marco Bonvini and Alberto Leva. Object-oriented sub-zonal models for energy-related building simulation. In *8th International Modelica Conference, Dresden (Germany)*, March 2011.

[6] Marco Bonvini and Alberto Leva. Object-oriented sub-zonal modelling for efficient energy-related building simulation. *Mathematical and Computer Modelling of Dynamical Systems*, 2011.

[7] Yunus Çengel. *Termodinamica e trasmissione del calore.* Zanichelli, 2006.

[8] Standard Performance Evaluation Corporation. SPECpower_ssj2008 Benchmark. `http://www.spec.org/power_ssj2008/`.

[9] Paolo Cremonesi and Andrea Sansottera. Cooling-aware workload placement with performance constraints. In *InfQ Workshop, Lipari (Italy)*, June 2011.

[10] STULZ GmbH. *CyberAir 2 brochure.*

[11] Dell Inc. *PowerEdge rack brochure.*

[12] Dell Inc. *PowerEdge M1000e technical guidebook.*

[13] Dell Inc. *PowerEdge M610 technical guide.*

[14] Dell Inc. *PowerEdge servers portfolio guide.*

[15] Kailash C. Karki, Amir Radmehr, and Suhas V. Patankar. Use of computational fluid dynamics for calculating flow rates through perforated tiles in raised-floor data centers. *International Journal of Heating, Ventilation, Air-Conditioning, and Refrigeration Research*, 9(2):153–166, April 2011.

[16] Stefan La Rocca. Cooling-aware data-center consolidation. Master's thesis, Politecnico di Milano, 2010/2011.

[17] David Moss. Guidelines for assessing power and cooling requirements in the data center. *Dell Power Solutions*, August 2005.

[18] Chandrakant D. Patel, Cullen E. Bash, and Christian Belady. Computational fluid dynamics modeling of high compute density data centers to assure system inlet air specifications. In *IPACK '01, The Pacific Rim/ASME International Electronic Packaging Technical Conference and Exhibition*, Kuai (Hawaii, USA), July 2001.

[19] Mark Rogoway. Facebook unveils Prineville data center design, pledges to collaborate on efficiency. `http://www.oregonlive.com/business/index.ssf/2011/04/facebook_post.html`.

[20] Erica Zavaglio. Quasi3d models for energy-related simulation of air volumes in buildings. Master's thesis, Politecnico di Milano, 2010/2011.