Gran Sasso Science Institute

Doctoral Thesis

Energy Efficiency in Large Scale Information Retrieval Systems

Author:
Matteo Catena

Supervisor:
Dr.-Ing. Nicola Tonellotto

2017

Declaration of Authorship

I, Matteo Catena, declare that this thesis has been composed by myself and the presented work is my own under the guidance of my supervisor Dr.-Ing. Nicola Tonellotto. Moreover, Chapter 3 is partially based on [25] co-authored with Dr.-Ing. Nicola Tonellotto. Chapter 4 is based on [24] co-authored with Dr. Craig Macdonald and Dr.-Ing. Nicola Tonellotto. Chapter 5 is based on [26] co-authored with Dr.-Ing. Nicola Tonellotto. Finally, Chapter 6 is based on [13] co-authored with Dr. Roi Blanco and Dr.-Ing. Nicola Tonellotto.

Abstract

Web search engines are large scale information retrieval systems, which provide easy access to information on the Web. High performance query processing is fundamental for the success of such systems. In fact, web search engines can receive billions of queries per day. Additionally, the issuing users are often impatient and expect sub-second response times to their queries (e.g., 500 ms). For such reasons, search companies adopt distributed query processing strategies to cope with huge volumes of incoming queries and to provide sub-second response times.

Web search engines perform distributed query processing on computer clusters composed by thousands of computers and hosted in large data centers. While data center facilities enable large-scale online services, they also raise economical and environmental concerns. Therefore, an important problem to address is how to reduce the energy expenditure of data centers. Moreover, another problem to tackle is how to reduce carbon dioxide emissions and the negative impact of the data centers on the environment.

A large part of the energy consumption of a data center could be accounted to inefficiencies in its cooling and power supply systems. However, search companies already adopt state-of-the art techniques to reduce the energy wastage of such systems, leaving little room for more improvements in those areas. Therefore, new approaches are necessary to mitigate the environmental impact and the energy expenditure of web search engines.

One option is to reduce the energy consumption of computing resources to mitigate the energy expenditure and carbon footprint of a search company. In particular, reducing the energy consumption of CPUs represents an attractive venue for web search engines. Currently, CPU cores frequencies are typically managed by operating system components, called frequency governors. We propose to delegate the CPU power management from the OS frequency governors to the query processing application. Such search engine-specific governors can reduce up to 24% a server power consumption, with only limited (but uncontrollable) drawbacks in the quality of search results with respect to a system running at maximum CPU frequency.

Since users can hardly notice response times that are faster than their expectations we advise that web search engine should not process queries faster than user expectations and, consequently, we propose the Predictive Energy Saving Online Scheduling (PESOS) algorithm, to select the most appropriate CPU frequency to process a query by its deadline, on a per-core basis. PESOS can reduce the CPU energy consumption of a query processing server from 24% up to 48% when compared to a high performance system running at maximum CPU core frequency.

To reduce the carbon footprint of web search engines, another option consists in using green energy to partially power their data centers. Stemming from these observations, we propose a new query forwarding algorithm that exploits both the green energy sources available at different data centers and the differences in market energy prices. The proposed solution maintains a high query throughput, while reducing by up to 25% the energy operational costs of multi-center search engines.

Acknowledgements

I would like to thank Dr.-Ing. Nicola Tonellotto for having been a rigorous yet enthusiastic thesis advisor. I would also like to thank Dr. Craig Macdonald and Dr. Roi Blanco for the pleasant and fruitful collaborations.

I want to thank all the people from the High Performance Computing Laboratory of Pisa, and all those who visited our group in these years. Thank you all for the useful conversations, for the good time spent together, and for making our workplace relaxing and enjoyable.

I want to thank Cristina for being a great co-worker and, more importantly, a great friend. Settling in from L'Aquila to Pisa would have been much harder without you. For the same reason, I thank Alejandro, Daniele, Diego & Laura, Giacomo, and Paola. Thank you for the fun time spent together and for the countless beers at La Torre del Luppolo.

Finally, I want to thank all the students of the XXIX cycle at the Gran Sasso Science Institute. Thank you for sharing all the fun and the stress. It has been quite an adventure.

All'amato me stesso.

Contents

List of Figures

List of Tables

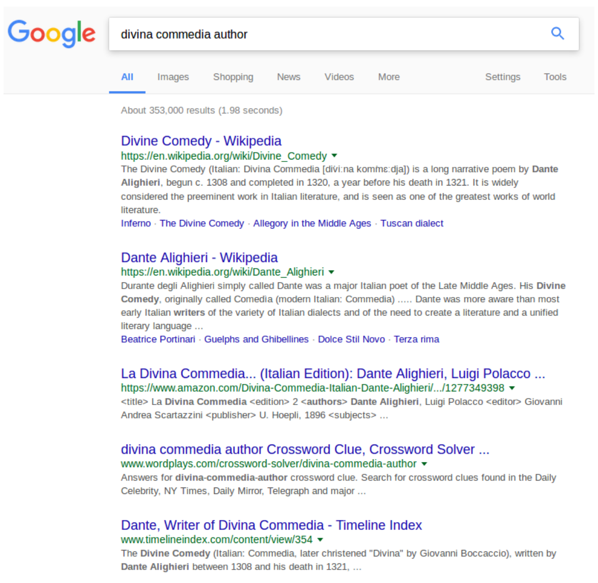
Chapter 1

Introduction

Information Retrieval (IR) is an area of Computer Science which focuses on the representation, storage, and organization of information items such as web pages, images, videos, etc. Its aim is to provide the users with easy access to information of their interest [7]. Information Retrieval finds a practical application in the development of web search engines.

Nowadays, web search engines are a fundamental tool in people's life, mostly due to their widespread usage and to the enormous size of the Web. Indeed, people daily use search engines for many purposes, such as looking for information on the Web, navigate to websites, or purchasing products and services online. Without search engines, users would have to personally skim through a myriad of web pages to find what they need. Surfing the Web would be a far less pleasant activity.

The purpose of web search engines is to make a collection of web pages searchable by their users. Users express their information needs using *queries* i.e., lists of keywords or query terms. For instance, a user interested in knowing "Who is the author of the Divina Commedia?" would probably issue a query like `"divina commedia author"`. In response to such query, the search engine retrieves the web pages containing the query terms, and it ranks them according to their estimated relevance, i.e., the ability to satisfy the information need of the user. As shown in Figure 1.1, the ranked list of results is then presented to the user by reporting the title of the web pages, short summaries, and links to the original sources.



About 353,000 results (1.98 seconds)

Divine Comedy - Wikipedia
https://en.wikipedia.org/wiki/Divine_Comedy
The Divine Comedy (Italian: Divina Commedia [diˈviːna komˈmɛːdja]) is a long narrative poem by **Dante Alighieri**, begun c. 1308 and completed in 1320, a year before his death in 1321. It is widely considered the preeminent work in Italian literature, and is seen as one of the greatest works of world literature.
Inferno · The Divine Comedy · Allegory in the Middle Ages · Tuscan dialect

Dante Alighieri - Wikipedia
https://en.wikipedia.org/wiki/Dante_Alighieri
Durante degli Alighieri simply called Dante was a major Italian poet of the Late Middle Ages. His **Divine Comedy**, originally called Comedia (modern Italian: Commedia) ..... Dante was more aware than most early Italian **writers** of the variety of Italian dialects and of the need to create a literature and a unified literary language ...
Beatrice Portinari · Guelphs and Ghibellines · Dolce Stil Novo · Terza rima

La Divina Commedia... (Italian Edition): Dante Alighieri, Luigi Polacco ...
https://www.amazon.com/Divina-Commedia-Italian-Dante-Alighieri/.../1277349398
<title> La **Divina Commedia** <edition> 2 <**authors**> **Dante Alighieri**, Luigi Polacco <editor> Giovanni Andrea Scartazzini <publisher> U. Hoepli, 1896 <subjects> ...

divina commedia author Crossword Clue, Crossword Solver ...
www.wordplays.com/crossword-solver/divina-commedia-author
Answers for **divina-commedia-author** crossword clue. Search for crossword clues found in the Daily Celebrity, NY Times, Daily Mirror, Telegraph and major ...

Dante, Writer of Divina Commedia - Timeline Index
www.timelineindex.com/content/view/354
The **Divine Comedy** (Italian: Commedia, later christened "Divina" by Giovanni Boccaccio), written by **Dante Alighieri** between 1308 and his death in 1321, ...

FIGURE 1.1: A search engine's results for the query `"divina commedia author"`, as shown to the issuing user.

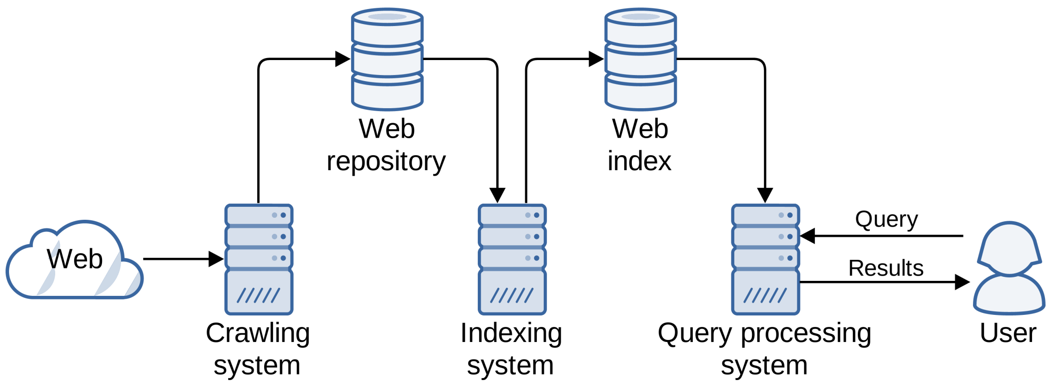


Figure 1.2: The architecture of a web search engine composed by the crawling, indexing and query processing subsystems.

To provide a list of results for the query `"divina commedia author"`, the search engine must have firstly gathered a *collection* of web pages. Within the search engine, this task is performed by the *crawling* subsystem. This subsystem is responsible to discover and fetch web pages that may be relevant to user queries, and to store them in a repository (i.e., the document collection). While the crawling subsystem gathers web pages, the search engine must make these searchable by using query terms. To this end, the *indexing* subsystem prepares a searchable index of the document collection. Such index permits to quickly locate the web pages matching query keywords. Finally, the *query processing* subsystem answers queries by accessing this index and providing the issuing user with a ranked list of results. As shown in Figure 1.2, a functional web search engine is composed by at least the crawling, indexing, and query processing subsystems, which operate in pipeline [20].

High performance crawling, indexing, and query processing subsystems are fundamental for the success of a web search engine. For instance, the number of individual web pages is estimated to grow by several billions every day [4]. To be effective, a search service should allow its user to search and find these new contents as soon as possible. To this end, the crawling subsystem must be able to quickly find and fetch newly created web pages, and the indexing subsystem must promptly index them. However, a good coverage of web contents it is not sufficient, alone, to satisfy users and to guarantee the success of a search engine. High performance query processing is important as well, since web search engine can receive billions of queries per day [48]. Additionally, their users are often impatient and expect sub-second response times to their queries (e.g., 500 ms). Indeed, users become less engaged [5] or migrate to other search services [97] when a search engine fails to provide fast responses to queries.

For the aforementioned reasons, web search engines deploy the architecture described in Figure 1.2 over large clusters constituted by tens of thousands of computers [11]. Such clusters provide the computing and storage capabilities required to quickly crawl and index billions of web pages [111]. Also, clusters are used to process queries in a distributed manner to cope with the huge volume of queries received by web search engines and to provide sub-second response times [33].

Big companies like Google, Microsoft, or Yahoo!, started building *data centers* to house their computer clusters. A data center is a facility which hosts large computer systems together with the infrastructures necessary for their proper functioning [10]. For instance, such associated infrastructures include the telecommunication system, i.e., the networking components which are necessary to inter-connect the computers of a cluster and to connect the clusters to the Internet. Other important infrastructures are the power supply system, which deliver electricity to the clusters, and the thermal cooling system, which reduces the heat generated by the data center's hardware.

A search engine's data center can host thousands of servers for the indexing and crawling subsystem to efficiently perform their tasks [54]. However, ten times more servers can be hosted in the same data center for the query processing subsystem to quickly answer huge volumes of queries. Nevertheless, a single data center may be not enough to this end. In fact, the query latencies experienced by a user do not depend only by the time required to process her queries. Network latencies must be taken into account as well [5, 97]. These consist in the amount of time required to send a query from the user to the search engine data center and to send the query results from the data center back to the issuing user. Since network latencies increase with the geographical distance between the user and the data center [57], web search engines distribute their infrastructures and operations across several, geographically distant data centers. User queries are then processed in the data center which is closest to the issuing user to yield sufficiently low query latencies [8]. Web search engines composed by multiple data centers are sometimes referred to as *multi-center web search engines*.

1.1 Energy-related challenges in data centers

Similar to web search engines, many other online services are hosted by large-scale data centers. Some examples are given by social network, map, web mail and cloud computing services. Thanks to their processing and storage capabilities, data centers contribute to the success of such services but, at the same time, they also raise energy-related concerns. Indeed, a large-scale data center – like those used by web search engines – can draw tens of Megawatts of electricity to operate [51]. Worldwide, data centers were estimated in 2012 to use the equivalent power output of 30 nuclear power plants, about 30 Gigawatts[1] of electricity [46]. Such energy consumption raise two different kinds of challenges: *environmental* and *economical*.

Depending on the energy source, producing and consuming electricity can involve the emission of carbon dioxide, which is the main cause of global warming due to the greenhouse effect. In 2007, the Information and Communication Technology (ICT) sector has been reported to be responsible for roughly 2% of global carbon emissions, with general purpose data centers accounting for 14% of the ICT footprint. These emission levels were projected to more than double by 2020 [102]. Carbon emissions can be reduced by using *green energy* i.e., energy that comes from resources which are renewable and do not emit carbon dioxide, such as sunlight and wind. In fact, several web companies report to use green energy to partially power their data centers [38, 50, 81]. However, solar and wind energy are not always available due to their susceptibility to weather conditions. As a consequence, data centers still have to rely on *brown energy* to function, i.e., energy which is produced using polluting resources like carbon or oil. Therefore, an important problem to tackle is *how to reduce the negative impact of the data centers on the environment*.

Additionally, high energy consumption represent not only an environmental issue, but also an economical one. In fact, the electricity expenditure of a data center can exceed its original investment cost, accounting for over 10% of its total ownership cost [63, 106]. While data center operators can purchase electricity at wholesale rates

[1]This is approximately the same amount of energy required for 25 time travels with the DeLorean from Back to the Future.

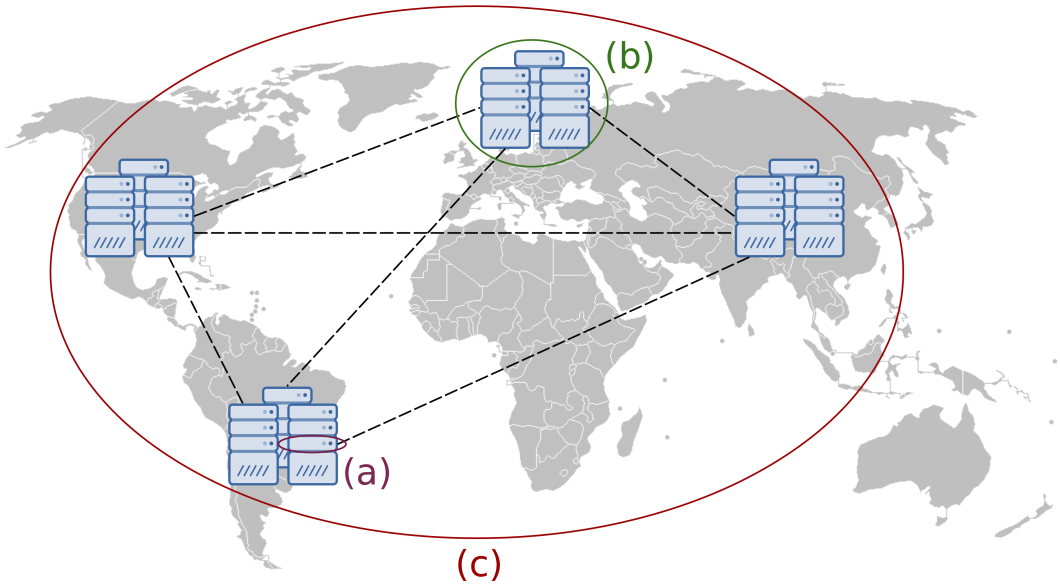


FIGURE 1.3: A web search engine architecture subdivided into the (a) intra-server, (b) intra-data center, and (c) inter-data centers levels.

to obtain better pricing, running a large-scale data center has been estimated to cost 9 millions US dollars per year in terms of energy expenditure [51]. Therefore, another problem to address is *how to reduce the energy expenditure of data centers*.

Obviously, a possible solution to these problems consists in designing more energy-efficient data centers, which consume less energy and, consequently, pollute and cost less. In the past, a large part of the energy consumption of a data center could be accounted to inefficiencies in its cooling and power supply systems. However, modern data centers have largely reduced the energy wastage of those infrastructures, leaving little room for more improvements in those areas. Indeed, the energy consumption of a state-of-the-art data center would be reduced by less than 24% if the overheads in its cooling and power supply systems were eliminated [10].

1.2 Energy management in web search engines

Being composed by multiple, large-scale data centers, web search engines are affected by the energy-related problems we have identified in Section 1.1. In their data centers, several search companies already adopt state-of-the art techniques to reduce the inefficiencies of cooling and power supply systems [49, 82]. Therefore, new approaches are necessary to mitigate the environmental impact and the energy expenditure of web search engines.

In this work, we primarily focus on reducing the energy consumption and expenditure, and the carbon footprint related to the query processing operations carried out by large-scale search systems. In the architecture of a multi-center search engine, we identify three levels at which it is possible to intervene: (a) the *intra-server*, (b) the *intra-data center*, and (c) the *inter-data centers* levels (see Figure 1.3). For each level, we will briefly illustrate some approaches that can be used to mitigate the energy-related problems of the query processing system, together with their limitations. A more detailed discussion will be provided in Chapter 2.

The approaches operating at the intra-server level aim at reducing the energy expenditure of the hardware components of single servers within a web search engine.

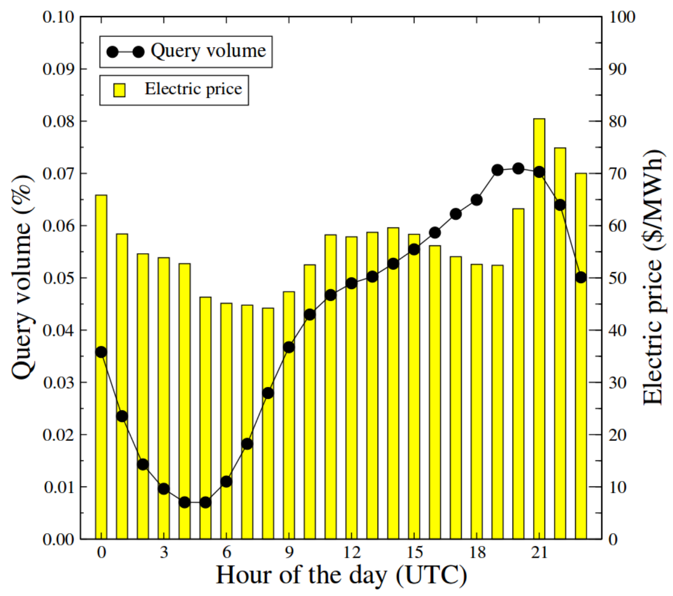


FIGURE 1.4: Yahoo! hourly query traffic volume distribution, and New York's hourly variation in electricity price (picture from [96]).

For instance, it is possible to reduce the CPU energy consumption of a server thanks to Dynamic Frequency and Voltage Scaling (DVFS) technologies [99]. These techniques allow to adjust the frequency and voltage at which the CPU cores operate, trading off performance for power consumption. In fact, higher core frequencies mean faster computations but higher power consumption, while lower frequencies lead to slower computations but reduced power consumption. DVFS technologies can be exploited to reduce the CPU energy consumption of a query processing server, thus reducing the energy expenditure of the whole server. However, carefulness is required when reducing the operating frequency of the CPU cores since low frequencies entail longer processing times that may be unacceptable for the users of a search engine.

Instead, the intra-data center level is composed by the computing resources within an individual data center, i.e., by its computer clusters. The operations of these resources can be coordinated to reduce the energy expenditure of a single data center within a multi-center web search engine. For example, it is possible to reduce the electricity consumption of a data center by exploiting the daily variations in its query workload. In fact, as shown in Figure 1.4, users send more queries between the morning and the evening than during the night. Therefore, the number of active query processing servers in a data center can be reduced when workload is scarce, by placing the idle servers in an inactive, energy saving state. Conversely, servers can be re-activated when the query workload becomes more intense [42]. However, queries must still be processed while providing the low latencies expected by the users. Therefore, it is important to keep an adequate number of servers active to provide acceptable query processing times.

Finally, the inter-data center level consists in all the data center which compose the web search engine. At this level, the operations of multiple data centers can be coordinated to reduce the overall energy expenditure or carbon footprint of the search engine. For instance, it is possible to reduce the operational cost of a multi-center search engine by exploiting spatial and temporal variations of electricity prices. In fact, the cost of electricity varies from country to country [91]. Additionally, as shown in Figure 1.4, the electricity price fluctuates during the day, due to supply/demand factors. *Query forwarding* has been proposed as a mean to reduce the energy expenditure of search engines whose data centers are located in different countries [61, 101]. The main idea is to dispatch queries from the data center that firstly received the requests to a different one. In order to reduce the energy expenditure, query forwarding aims at shifting the query load towards those data centers which incur in the lowest energy prince in that moment. However, it is important to not overload a site with queries coming from other data centers. Indeed, exceeding the processing capability of a data center entails longer query response times, which may be unacceptable for the users of the search engine. Longer response times can also occur because of network latencies between data centers, which must be taken into account when forwarding queries.

1.3 Summary of contributions

In this work, we focus on reducing the energy consumption and expenditure, and the carbon footprint of the query processing operations in multi-center search engines. To this end, we tackle two main research questions: (a) is it possible to reduce the energy consumption of search servers? and (b) is it possible to reduce the carbon footprint and energy expenditure of a multi-center search engine?

Regarding question (a), we intervene at the intra-server level to reduce the energy consumption of query processing servers, by leveraging DVFS technologies. Currently, CPU cores frequencies are managed by operating system (OS) components, called *frequency governors* [17, 103]. A popular policy in such governors is to throttle a core frequency according to its utilization, selecting a high frequency when a core is highly utilized and a lower one when the core is lowly utilized. However, core utilization-based policies have no mean to impose a required latency on a query processing server, since they are devised for general-purpose computing. As a result, latency violations can occur when core utilization-based policies are used in a web search engine [71].

For such reason, in “Load-sensitive CPU Power Management for Web Search Engines” we propose to delegate the CPU power management from the OS frequency governors to search engine-specific ones [24]. In fact, the operating system misses domain-specific information on the search engine software and its interaction with the incoming user queries. We advocate that this information can help to improve the energy efficiency of a search engine. Therefore, we propose search engine-specific frequency governors that can better adapt to varying query workloads. Our governors leverage domain-specific information such as the query processing server utilization and load. By exploiting such additional knowledge, the proposed governors can appropriately throttle the frequency of all the CPU cores thereby reducing the power consumption of a query processing server. Experiments are conducted upon the TREC ClueWeb09B corpus [67] and the query stream from the MSN 2006 query log [83]. Results show that the knowledge of the query processing server utilization and load facilitates a more refined control of the CPU to achieve power savings. In fact, our search engine-specific governors can reduce up to 24% a server power consumption, with limited (but uncontrollable) drawbacks in the quality of search results with respect to a system running at maximum CPU frequency.

Another important aspect that can be exploited to reduce the energy consumption at the intra-server level is the fact that users can hardly notice response times that are faster than their expectations (e.g., below 500 ms) [5, 97]. Therefore, we advise that web search engine should not process queries faster than user expectations, and we propose the Predictive Energy Saving Online Scheduling (PESOS) algorithm in

“Energy-efficient Query Processing in Web Search Engines” [26]. PESOS selects the most appropriate CPU frequency to process a query by its deadline, on a per-core basis. PESOS considers the latency requirement of queries as an explicit parameter, and it tries to process queries no faster than required. In doing so, the CPU energy consumption is reduced while respecting the query latency constraints. PESOS bases its decision on query efficiency predictors, which are techniques to estimate the processing volume and processing time of a query before its execution [75]. We experimentally evaluate PESOS upon the TREC ClueWeb09B collection and the MSN 2006 query log. Depending on the required latency, results show that PESOS can reduce the CPU energy consumption of a query processing server from 24% up to 48% when compared to a high performance system running at maximum CPU core frequency. Also, PESOS outperforms the best approach presented in [24] with a 20% energy saving, while the competitor requires a fine parameter tuning and it may incurs in uncontrollable latency violations. The techniques illustrated in [24, 26] operate at the intra-server level, but they can also be deployed at the inter-data center level (i.e., on a search cluster) as they are completely decentralized.

Relatively to question (b), we move our attention at the inter-data center level, to reduce the energy expenditure and carbon footprint of a multi-center web search engines. In “Exploiting Green Energy to Reduce the Operational Costs of Multi-Center Web Search Engines”, we tackle the problem of targeting the usage of green energy to minimize the expenditure of running a multi-center web search engine [13]. For this purpose, we propose a new query forwarding algorithm that exploits both the green energy sources available at different sites and the differences in market energy prices. The problem of exploiting green/brown energy to reduce costs when forwarding queries is modeled as a Minimum Cost Flow Problem. The model takes into account the different and limited processing capacities of data centers, query response time constraints and communication latencies among sites. We evaluate the proposed algorithm using workloads obtained from the Yahoo search engine together with realistic electricity price data. Our experimental results show that the proposed solution maintains a high query throughput, while reducing by up to 25% the energy operational costs of multi-center search engines. Moreover, our algorithm can reduce the brown energy consumption by almost 6% when energy efficient servers are used in the data centers of a search engine, thus reducing its carbon footprint.

1.4 Organization of the thesis

The rest of this thesis is structured as follows. In Chapter 2 we provide some background regarding query processing, and we illustrate related work on energy management techniques for web search engines. In Chapter 3 we propose an experimental setting to measure the energy consumed to process queries, and we identify opportunities for web search engines to reduce their energy consumption. In Chapter 4 we introduce and experimentally evaluate our search-engine specific governors to control CPUs frequencies in search servers, while in Chapter 5 we propose PESOS and experimentally quantify its provided energy savings. In Chapter 6 we introduce and evaluate our query forwarding algorithm for reducing the energy expenditure and carbon footprint of multi-center web search engines. In Chapter 7 we sum up the contributions of this thesis and draw the final conclusions and future directions.

Chapter 2

Background

In this thesis, we will focus on improving the energy efficiency of the query processing subsystem. Therefore, in this chapter we will briefly discuss background and related works inherently to the query processing activities. The chapter is organized as follows. Firstly, we will provide a brief overview of the query processing subsystem when deployed on a single search server, on a search cluster, and on multiple data centers. Then, we will discuss which are the available options for improving the energy efficiency of query processing subsystem within web search engines.

2.1 Query processing

As shown in Figure 2.1, the query processing subsystem can be divided into three layers, with the higher layers relying on the functionalities of the lower ones. From bottom to top, these layers represent the query processing activities carried on by a) a single search server, b) a cluster of search servers within a data center, and c) multiple data centers. In the following, we will first discuss about single search servers (Section 2.1.1), then we will examine search clusters (Section 2.1.2), and finally we will talk about multiple search sites query processing (Section 2.1.3).

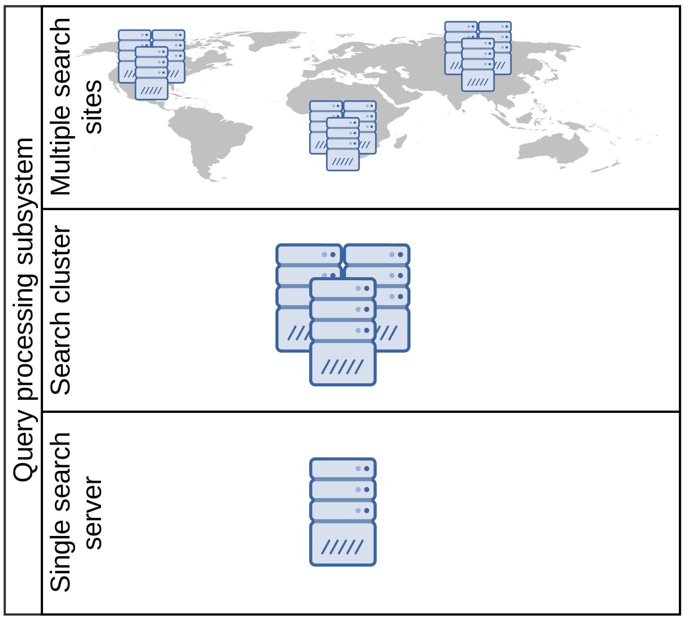


Figure 2.1: The query processing subsystem deployed onto three different layers: on a single search server, on a search cluster, and on multiple search sites.

2.1.1 Query processing on a search server

The most basic task performed by the query processing subsystem is to retrieve, from the document collection, all the Web pages containing a query's terms. To this end, the subsystem could inspect the textual content of all the web pages in the collection, perhaps using a string matching algorithm, to check which page contains at least one of the query terms. However, such naïve approach would be too time consuming due to the huge size of a typical web collection, which can contain billions of documents [111].

A more efficient approach to this task is to use an *inverted index* [7, 110]. The inverted index is a data structure which associates, to each term appearing in the document collection, a list of the pages which contain it. For each of such pages, the list contains a *posting* which stores, at the very least, the document identifier (docid), i.e., a natural number which uniquely identify the web page within the document collection. An example of a portion of an inverted index is reported in Figure 2.2. In the example, the term "dog" is contained in the documents with id 1 and 2, and the term "cat" is contained in the documents with id 1, 3, and 8.

For efficiency reasons, the inverted index is typically compressed to fit, as much as possible, in the main memory of a search server [18, 32]. Since the posting lists can be seen as lists of integers, integer compression algorithms are used to encode them [23, 65, 107]. Many of these algorithms are better at compressing small integers than large ones. Therefore, posting lists can be stored in the inverted index with their document identifiers sorted in ascending order. This permits to represent the identifiers by using *d-gaps*, i.e., the difference between an identifier and its preceding one in the posting list. Since d-gaps are smaller than the original document identifiers, they can be compressed more efficiently resulting in smaller index sizes [110].

It is easy to see how the inverted index permits to efficiently find all the pages in which a query term appears. In fact, the query processing subsystem needs just to access the posting list associated to that query term within the inverted index. Then, by traversing the list and decompressing its postings, the query processing subsystem can get the identifiers of the all the web pages in which that particular query term occurs. If a query is composed by more than one term, it suffices to intersect their posting lists to find those pages which contain all the query terms. Similarly, performing the union of the posting lists returns all the web pages containing at least one of the query terms.

While the inverted index permits to quickly identify the web pages matching a query, the number of such documents can easily exceed the quantity of documents that a human can analyze. Therefore, the query processing system usually retrieves from the inverted index only the top K results according to some simple ranking function f [77]. Let d be a document and q be a query, the function f associates to the pair (d, q) a *relevance score*, i.e., a real number which indicates the relevance of d with respect to q. The higher is the relevance score, the more a document is

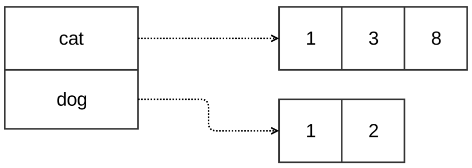


FIGURE 2.2: An example of a portion of an inverted index.

deemed to be relevant for a query. Ranking functions compute the relevance score by considering several parameters, such as the number of occurrences of a query term within the web page, the term popularity within the document collection, and the web page length [95]. The relevance score is often a linear combination of the relevance scores computed with respect to the single query terms. In other words, if the query q is composed by n terms $\{t_1, \ldots, t_n\}$, then $f(d, q) = \sum_{i=1}^{n} f(d, t_i)$.

For a given query, the top K results are the matching web pages with the K largest relevance scores according to the f function[1]. Clearly, to identify the top K pages for a query, all the matching documents must be scored by traversing the posting lists associated to the query terms[2]. Several strategies exist to traverse the posting lists [105]. For instance, in a term-at-a-time (TAAT) strategy, the posting lists of the query terms are processed one at a time accumulating the score of each document in a separate data structure. On the contrary, the document-at-a-time (DAAT) strategy simultaneously processes all the query term posting lists, keeping them aligned by document identifier. In other words, DAAT firstly considers the contributions of all the query terms to a document's relevance score and, only when the final score is calculated, it starts evaluating the next document matching the query. In this way, DAAT needs to maintain only K (top) scores during query processing, for instance by relying on a min-heap data structure. On the contrary, TAAT has to maintain an accumulator for each document matching the query, resulting in an larger memory footprint than DAAT.

To exhaustively score all the documents which match a query can be very inefficient with both TAAT and DAAT strategies. However, DAAT is more amenable than TAAT to *dynamic pruning* optimizations. These techniques reduce the query processing time by skipping the evaluation of documents which are unlikely to belong to the top K results for a query. The postings of such documents are not decompressed neither scored, hence reducing the amount of time required for query processing. Popular dynamic pruning techniques are MaxScore [105], WAND [16], and Block-Max WAND [35]. All these strategies augment the inverted index by storing for each term its maximum score contribution, according to the ranking function adopted by the search engine. This allows skipping large segments of the posting lists if they only contain documents whose sum of maximum scores is smaller than the "threshold", i.e., the score of the least relevant document in the top K up to that point.

In particular, the MaxScore algorithm partition query terms in "essentials" and "non-essentials". At the beginning of the query processing, MaxScore considers all the query terms to be essential. As the query processing proceeds, documents are inserted into the top K results and the threshold increases. Subsequently, a term becomes non-essential as soon as the threshold exceeds its maximum score. Documents containing only the non-essential term are then skipped, as they cannot belong to the top K results. Additional terms can be demoted to be non-essential as the query processing continues. This happens if the threshold becomes greater than the sum of a term's and the non-essential terms' maximum scores. As a consequence, more documents can be skipped while MaxScore evaluates only the documents containing at least one essential query term.

Differently from MaxScore, the WAND algorithm works by identifying a "pivot" document at each of its iterations. The pivot is a document which has the potential to enter the top K results, i.e., the sum of the maximum scores of its terms are greater than the threshold at that point. At the same time, the pivot document

[1]In web search engines, top K results are usually re-ranked by machine learning systems [70, 74].

[2]Postings are processed from the smallest to the largest docid, if the posting lists are sorted by document identifier (for instance, to enable d-gap compression).

has the smallest document identifier among the postings traversed by WAND at that iteration. Therefore, WAND can safely prune the evaluation of documents whose identifier is smaller than the pivot's, as they cannot enter into the top K results.

Finally, Block-Max WAND is an extension of WAND which identifies and exploits the pivot document in the same way. However, Block-Max WAND structures the index into blocks. For each block, Block-Max WAND maintain its maximum score contribution. This information is used to efficiently access the index blocks, in order to decide whether their documents should be processed or not, as they will never enter the list of top scoring results. Since block maximum scores are tighter than term maximum scores, much more documents will be pruned by Block-Max WAND with respect to WAND.

2.1.2 Query processing on a search cluster

Billions of queries are typed by users every day [48]. As a consequence, a search engine can receive thousands of queries per second, especially when users are more active, e.g., at daytime [98]. The query throughput defines how many query per second can be processed by the query processing subsystem. It is fundamental for a search engine to operate at high query throughput, ideally matching the query arrival rate. Contrarily, the search engine would not be able to process queries as the same pace of their arrival, resulting in a system overload which will negatively impact its performance. Clearly, a single search server cannot be sufficient to deal with such huge amount of queries. Therefore, the query processing subsystem is usually deployed on a cluster of servers which can adopt a replicated and a distributed architecture.

In the replicated architecture, each cluster's server holds a *replica* of the inverted index [42, 75]. Servers operate in parallel, processing different queries at the same time hence increasing the search engine throughput. When a user issues a query, it is first received by a dedicated server, the *query broker*, which routes the query on a search server. Once the search server has computed the query results, these are collected by the broker and sent back to the issuing user.

However, a web search engine must provide not only a high query throughput but also low query latency. From the user perspective, query latency is the amount of time elapsing between issuing the query and receiving its result. One way to reduce latencies is to reduce the query processing times. To this end, the index can be partitioned into smaller *shards*. In fact, query processing times increase with the posting lists' lengths, since more postings need to be traversed, decompressed, and scored [85]. Therefore, index partitioning aims at keeping the posting lists short so that query processing times are reduced. For instance, document-based partitioning assign different documents to different shards, such that each shard can act as an independent inverted index [6, 72, 73].

After partitioning, index shards are assigned to different search servers. As shown in Figure 2.3, incoming queries are first received by the broker which dispatches them to every search server. Each server computes the query results on its shard independently from the others. These partial results are sent back to the broker, which aggregates the results coming from different search servers [11, 33]. The aggregated results are the same that would be provided by a single inverted index but query processing times are reduced, since the computation is now distributed across several servers. Once the query broker has collected and aggregated the partial results coming from the shards, the final results can be sent back to the issuing user (see Figure 2.3).

By combining these replicated and distributed architectures, it is possible to improve both the query throughput and latency of a search system [11, 33]. In fact,

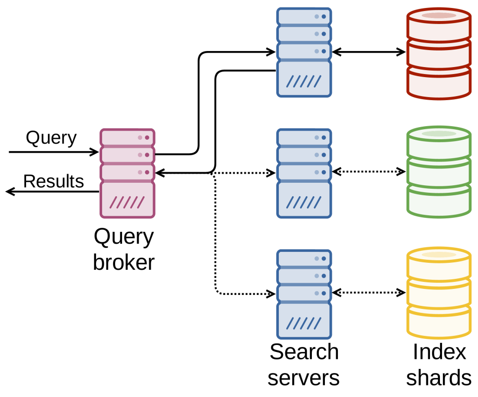


Figure 2.3: A distributed query processing subsystem with three index shards.

the inverted index can be firstly partitioned into multiple shards, to reduce the query processing time and latency. Then, each shard can be replicated several times on different servers, each hosting a different shard. In this way, different queries can be processed at the same time by different ensembles of shard replicas, improving the system throughput.

Finally, the query throughput and latency can be further improved by introducing caching mechanisms [66]. For instance, a result cache can be deployed on the query broker such that, when it receives a query, it can serve its results directly from the cache if they have been previously computed and memorized. In this case, the broker do not forward the query to the search cluster, avoiding expensive query processing and improving the system efficiency.

2.1.3 Query processing on multiple search centers

Rather than relying on a single data center, search engine can distribute query processing over multiple, geographically distant sites [21]. There are several reasons to adopt this architecture, such as better fault tolerance and business continuity. For instance, a data center could become unavailable due to a natural catastrophe or to a network malfunctioning. In such case, a multi-center search engine could still process user queries by leveraging its remaining data centers.

From an efficiency perspective, multiple centers can be conveniently exploited to reduce the query response times. This is possible by processing queries in the data center which is closest to the issuing user [8]. In fact, network latencies contributes to the query latencies experienced by the users [5, 97] and they increase with the geographical distance between the user and the data center [57]. By processing queries in the closest data center, network latencies are reduced thus lowering the search engine latencies.

Additionally, a multi-site architecture can be leveraged to manage an unexpected overload of queries [13, 61]. Indeed, the performance of a data center degrades if its processing capability is exceeded. Its load can be alleviated by query forwarding, i.e., by shifting the processing of a part of its queries towards other search sites. In this case, it is important to consider the load of the other sites to avoid the risk to overload additional data centers. Network latencies between sites must be taken into consideration as well, to ensure that query response times remain acceptable.

Table 2.1: Electric energy quantities, definitions and units of measurements.

Quantity	Symbol	Definition	Unit
Time	T	-	second (s)
Voltage	V	-	volt (V)
Current Intensity	I	-	ampere (A)
Power	P	$V \cdot I$	watt (W)
Energy	E	$P \cdot T$	joule (J) or watt-second (Ws)

2.2 Energy management in web search engines

Thanks to their processing and storage capabilities, data centers are fundamental for the success of web search engines. However, data centers entails also high energy consumption which raise economical and environmental concerns, as highlighted in Chapter 1. Indeed, a data center (and its computing resources) needs electric energy in order to function. Electricity is provided with a certain voltage and current intensity. The product of these two quantity determines the *power* consumption P of an electronic apparatus, while its *energy* consumption E is given by P multiplied by the amount of time the apparatus is functioning [40]. Units of measurements are summarized in Table 2.1.

In the past, a large part of the energy consumption of a data center was accounted to inefficiencies in its cooling and power supply systems. However, a careful design of the data center can drastically reduce the energy wastage of those infrastructures, and useful design guidelines are given in [10]. For instance, power losses in the uninterruptible power supplies (UPSes) and power distribution system can be largely reduced by adopting higher-efficiency gear, such as rotary UPSes. Instead, the energy efficiency of the cooling system can be improved by carefully handling the air flows within the data center, for instance by keeping cold air flows separated from the hot air flows exhausted by the servers. Additionally, almost no hardware equipment need to operate at 20°C. Therefore, temperatures within the data can be safely maintained around 30°C, reducing the stress on the cooling system and, consequently, its energy consumption. In fact, in mitigate climates, such temperatures can be obtained by exploiting the chill air coming from outside the data center.

Since web search engines have largely reduced the energy wastage of the cooling and power supply infrastructures, little room for more improvements in those areas is left [49, 82]. Indeed, the energy consumption of a state-of-the-art data center would be reduced by less than 24% if all the overheads in its cooling and power supply systems were eliminated [10]. Therefore, future gains in energy efficiency will need to come from improvements in the computing hardware and software.

Indeed, energy efficient computing is not a new research field and several approaches have been proposed to solve the energy-related problems of general-purpose data centers, like those used by data storage or batch processing services such as Dropbox or Elastic MapReduce [84, 86]. However, many of these techniques are inadequate for web search engines, as we will illustrate in the rest of this section. Therefore, new search engine-specific approaches are necessary to mitigate the environmental impact and the energy expenditure of such systems.

In the following, we will discuss related works regarding the energy efficiency of large scale Information Retrieval systems. The remaining of this Chapter will reflect the structure of Section 2.1 and the architectural levels we identified in Section 1.2.

In other words, we will first illustrate possible approaches to improve the energy efficiency of single search servers (intra-server energy management). Then, we will discuss the existing works aiming at improving the energy efficiency of search clusters (intra-data center energy management). Finally, we will talk about techniques for enhancing the energy efficiency of multi-center search engines (inter-data centers energy management).

2.2.1 Intra-server energy management

The power consumption P of a (generic) server can be roughly divided into two main components: 1) P_{static} which is the power required to keep the server idle but active, i.e., ready to perform computations, and 2) $P_{dynamic}$ which is the power required to actually do useful computations. Therefore, the overall power P consumed by a server is $P = P_{static} + P_{dynamic}$. While P_{static} is a constant, $P_{dynamic}$ linearly increases with the server utilization[3] [10]. Thus, a server reaches its peak power consumption when it is 100% utilized, i.e., when the server is constantly busy. However, it is not uncommon for a server to consume 50% of its peak power when it is idle without performing any demanding computations [12]. For this reason, Barroso and Hölzle report the data centers' need for *energy-proportional computing*, i.e., for hardware components with power consumption proportional to utilization [10, 12].

Typically, the energy consumption of a search server is dominated by its CPU. In fact, Barroso, Clidaras, and Hölzle [10, pages 78-79] show that "the CPU is the dominant energy consumer in servers, using two thirds of the energy at peak utilization and about 40% when (active) idle"[4]. In particular, the authors refers to the power usage of Google servers as the compute load varies from idle to full activity levels. For such reason, improving the energy efficiency of processors is an attractive research area for reducing the energy consumption of servers. Indeed, modern CPUs expose two energy saving mechanisms, namely *C-states* and *P-states*, to respectively reduce the CPU's contributions to the P_{idle} and P_{active} power consumption of a server.

C-states represent CPU cores idle states. They are denoted by an increasing numbers (e.g., C0, C1,...,Cn) and they are typically managed by the operating system [88]. C0 is the operative state in which a CPU core can perform computing tasks. When idle periods occur, i.e., when there are no computing tasks to perform, the core can enter one of the other C-states and become inoperative. When a core is inoperative it cannot perform any computing task and energy can be saved by turning off some CPU components, like clocks and caches. Deeper C-states correspond to greater energy savings but also to longer times for the core to return in its operative state C0. Idle states have been successfully exploited to reduce the energy consumption of servers [45, 79]. However, web search engines process a large and continuous stream of queries. As a result, search servers are rarely inactive and experience particularly short idle times. Consequently, there are little opportunities to exploit deep C-states, reducing the energy savings provided by the C-states in a web search engine system [71, 80][5].

Differently from C-states, P-states represent the frequencies available to the CPU cores when they are active. Indeed, when a CPU core is in the active C0 state, it can operate at different clock frequencies (e.g., 800 MHz, 1.6 GHz, 2.1 GHz,...). This is possible thanks to Dynamic Frequency and Voltage Scaling (DVFS) technologies [99]

[3]The server utilization is the percentage of time during which the server is busy performing computations.

[4]We confirm this finding in Appendix A.

[5]We confirm this finding in Appendix B.

which permit to adjust the frequency and voltage of a core to vary its performance and power consumption. In fact, higher core frequencies mean faster computations but higher power consumption. Vice versa, lower frequencies lead to slower computations and reduced power consumption. The various configurations of voltage and frequency available to the CPU cores are mapped to different P-states. Like for the C-states, P-states are denoted by an increasing numbers (e.g., `P0`, `P1`,...,`Pn`), where `P0` corresponds to the highest frequency and `Pn` to the lowest. Similar to the C-states, also the P-states are managed by the operating system through some predefined policies, sometimes referred to as *frequency governors* [17, 103]. For instance, the `intel_pstate` driver [103] controls the P-states on Linux systems[6] and can operate accordingly to two different governors, namely `performance` and `powersave`. The `performance` policy simply uses the highest frequency to process computing tasks. Instead, `powersave` adaptively selects the frequency for a core according to its utilization. When a core is highly utilized, `powersave` selects a high frequency. Conversely, it will select a lower frequency when the core is lowly utilized. However, Lo et al. argue that core utilization is a poor choice for managing the cores frequencies of search servers [71]. In fact, the authors report an increase of query response times when core utilization-based policies are used in a web search engine.

Since utilization-based controllers are inefficient in search servers, more specific approaches are necessary to effectively manage DVFS mechanisms. Du et al. propose the Dynamic Equal Sharing (DES) algorithm to keep the CPU power consumption of a search server within a certain power budget [37]. To this end, DES firstly assigns queries to cores in a round-robin fashion to balance their loads. Then, it dynamically distributes power among the cores according to their respective load. Finally, DES selects for each core the most appropriate operative frequency such that the power budget is respected and most of the queries are processed within their deadlines (e.g., 150 ms since their arrival). The authors evaluate DES by simulation and they find that DES can reduce by more than 40% the CPU dynamic power consumption of a lightly loaded search server, i.e., receiving 100 incoming queries per second. However, DES assumes that query processing can be terminated before its completion, i.e., queries can be partially evaluated and result quality can be degraded. Additionally, DES unrealistically assumes that the exact processing time of a query is known before its processing.

Rather then relying on traditional power saving mechanism, such as the DVFS technologies, several works have explored the possibility of using non-conventional hardware to implement energy efficient query processing. For instance, Janapa Reddi et al. investigate the possibility for a search engine to adopt low-power consuming CPUs, like those used in mobile devices [58]. In terms of processed queries per second over consumed Watts, the authors find that such CPUs are five times more efficient than server-class processors. However, they also report that such mobile processors negatively impact the search engine performance, as the query latencies increase and become more variable than what happens with server-class CPUs.

Instead of adopting a single class of processors, Ren et al. envision the possibility of using heterogeneous CPUs in search servers [94]. On the same chip, heterogeneous processors include few, high power consuming, fast cores and many, low power consuming, slow cores. The authors propose a scheduling algorithm that promotes the execution of long-running queries on the fast cores while most of the short-running queries complete their execution on the slower ones. The authors evaluate their approach by simulation, modeling the workload of the Bing search engine. They show

[6]`intel_pstate` is currently the default driver on Ubuntu distributions.

that their scheduling algorithm, along with heterogeneous processors, can improve the system throughput by up to 50% compared to a system which solely exploits homogeneous hardware. Noticeably, the proposed system would be more energy efficient than the baseline, as the two systems would consume the same amount of power.

A different line of research aims at leveraging Graphics Processing Units (GPUs) in the query processing subsystem. Ding et al. find that, by leveraging both CPU and GPU on a search server, it is possible to increase by three times the query throughput with respect to a server which exclusively relies on its CPU [36]. However, the authors believe that more work is necessary to understand whether GPUs can be cost-effective for query processing. Indeed, due to the high-power consumption of GPUs [90], the authors consider server-class CPUs to still provide better value than GPUs for what concerns search systems.

Finally, several works have explored the possibility of using Field Programmable Gate Arrays (FPGAs) within a search engine to improve its performance. Yan et al. implemented a query processing system which uses FPGAs and evaluates queries by accessing an inverted index hosted on flash memories [113]. In terms of query throughput, they found their system to be non competitive against a traditional one based on commodity hardware. The authors justify such negative result because of the limitations in the memory bandwidth of flash memories. If such limitations would be removed, the authors claim that the proposed FPGA-based system would be 2.4 times more efficient than a conventional system in terms of processed queries per second over consumed Watts. The energy efficiency of FPGAs is confirmed also by Chalamalasetti et al. They implemented a FPGA-based system for document filtering which provides a 23 times higher throughput than a conventional system, while consuming about half of its power [27]. Finally, Putnam et al. used FPGAs to implement the second stage of the Bing ranking framework [90]. While consuming just 10% more energy than a commodity-hardware based system, they improved the throughput of each search server by a factor of almost two. However, Putnam et al. also claim that it is challenging to program FPGAs, as they still require extensive hand-coding in register transfer level (RTL) and manual tuning.

Due to the challenges involved with the adoption of mobile processors, GPUs, and FPGAs, we will focus in this thesis on improving the energy efficiency of commodity hardware. In fact, commodity hardware has been already used to build successful, large scale search engines [11]. In particular, we will concentrate our attention on reducing the energy consumption of server-class CPUs, as they dominate the energy consumption of search servers.

2.2.2 Intra-data center energy management

In general-purpose data centers, it is possible to improve the energy efficiency of computer clusters by managing the servers as an ensemble. For instance, Ranganathan et al. studied the problem of managing a rack of servers which must respect a given power cap [93]. The authors find that a centralized solution is effective in enforcing the power budget constraint, actually reducing by 20% the system power consumption. In another work, Leverich and Kozyrakis studied how to recast the data layout and task distribution of a Hadoop cluster such to enable workload consolidation [68]. During periods of low load, workload consolidation permits to aggregate computing load onto few servers in order to power down the idle ones. Leverich and Kozyrakis find that running Hadoop clusters in fractional configurations can save up to 50% of energy consumption, trading off performance for energy efficiency. Raghavendra et al. propose to unify the benefits of power capping and workload consolidation. To this end, the authors devise a power management scheme which effectively coordinates both the mechanisms [92].

However, Barroso, Clidaras, and Hölzle report that power capping is not widely adopted in data centers which host latency-critical applications such as search engines [10]. In fact, search applications can become unstable if some servers unpredictably slow down due to power capping. This happens because search engines distribute query processing on several servers. In such distributed systems, the query processing time depends on the last server to finish its query processing subtask. Therefore, power capping may undesirably prolong query processing times by slowing down search servers. Similarly, workload consolidation is impractical for distributed search engines, since each server hosts a different index shard. As a consequence, it is not possible to turn off a server as each shard must be available for carrying out the query processing activities.

However, workload consolidation can be exploited when a search engine adopt a replicated rather than a distributed architecture. Recently, Freire et al. have proposed a self-adaptive model to manage the number of active search servers in a replicated search engine, while guaranteeing acceptable response times [42]. By exploiting the historical and current query loads, their model autonomously decides whether to activate a search server or put it in standby. The latter option permits to reduce the energy consumption of the system during low query loads, while the former permits to increase the system performance when the system faces a high query volume. The model is formulated as a decision problem which tries to optimize the power/latency trade off, by estimating future query arrival and service times. The authors validate their approach by simulation. Results show that the proposed model reduces by 33% the search engine energy consumption, with respect to a naïve baseline where search servers are always active. At the same time, the authors observe only little increments in query response time and small percentages of unanswered queries, i.e., queries that are not processed within an acceptable time since their arrival.

Lo et al. introduced PEGASUS, a feedback-based model that dynamically cap the CPUs power consumption of a distributed search engine [71]. Their approach trades off power savings for longer latencies that barely meet the query response time requirements under any query workload. Basically, PEGASUS constantly monitors the search engine latency and passes this value to a centralized rule engine. Depending on the observed latency, the rule engine decide whether to increase or decrease the CPUs performance by exploiting DVFS technologies. Experimenting on Google production cluster, the authors observe a 20% power consumption reduction and they estimate that a distributed version of PEGASUS could nearly double those savings.

Caching mechanisms can play a role for energy efficiency, too. For instance, Marin, Gil-Costa, and Gomez-Pantoja propose a caching mechanism for distributed search engines which can reduce energy consumption [78]. In their approach, the query broker records which shards have generated the top K results for a query. When the same query is received again by the system, it is propagated only to the nodes hosting the shards which produced the top K results, if these are not cached. While their work does not focus on energy consumption, the authors find that their approach reduces servers utilization and leads to energy savings. By mean of simulation, Marin, Gil-Costa, and Gomez-Pantoja find that their approach reduces the search engine power consumption by more than 30% with respect to classical caching techniques.

Sazoglu et al. observe how energy price and query volume vary during the day. Taking this aspect into account, they present a financial cost metric to measure the price of cache misses [96]. They define the monetary cost of processing a query as the product between the query processing time and the electricity price at the query arrival time. The authors use this metric to evaluate many state-of-the-art caching techniques: Static Dynamic Cache (SDC) [39] proves to have the lowest financial cost, when adapted to take into account query processing overheads [44]. The authors perform additional evaluations omitting the processing time factor from their financial cost metric, since search engines can limit the maximum time spent processing a single query [61]. In such case, the authors find that their financial cost metric approximate the hit rate one, if energy price exhibits only small variations over time. Additionally, they observe that the benefits provided by cost-aware strategies are more evident when there is a high variation in the query costs. Finally, the authors propose a novel caching whose admission policy takes into explicit account the query financial costs. This permits expensive queries to remain longer in the result cache, in order to reduce the search engine operational costs. By experimental evaluation, the authors show that their caching mechanism performs close to other state-of-the-art caching techniques, in term of cache hit rates and financial cost.

2.2.3 Inter-data centers energy management

To improve the energy efficiency of multi-center search engines, it has been proposed to leverage spatial and temporal variations in both the energy prices and query workloads. In fact, it has been observed that (a) energy price varies between different countries and it changes during the day, being more costly during daytime than at nighttime [91], and (b) query volumes fluctuate over time, with more active users at day than at night [98]. Due to time zone differences, a search engine's data center may experience a high workload and energy price in a certain moment, while other distant sites are underutilized and can use cheaper electricity. Thus, the first data center could forward its queries to the other sites to reduce its energy expenditure. However, network latencies have to be carefully considered, not to exceed acceptable query response times. Also, data centers have limited processing capacity. Therefore, it is not possible to forward too many queries towards a site.

Stemming from these considerations, Kayaaslan et al. investigated the possibility to dynamically shift the query workload among data centers by using query forwarding [61]. The authors model the problem as an online optimization one, which minimizes the total energy cost by forwarding queries among data centers while satisfying the performance constraints. The authors simulate a scenario where five geographically distributed data centers host the same inverted index and process real queries from the Yahoo! search engine. Results show that multi-center search engines can save up to 35% in energy expenditure, when compared to a system which always locally solves its incoming queries.

Inspired by the same observations on the variability of energy costs and query volumes, Teymorian, Frieder, and Maloof propose the Rank-Energy Selective Query forwarding (RESQ) algorithm [101]. The authors consider a scenario where different data centers hold different inverted indexes. A data center always processes its queries locally but it can also forward a query to other sites to gather more results. However, the available monetary budget for query processing is limited and query forwarding is considered to have a cost. Therefore, a data center forwards a query only towards those sites which (a) show low energy prices and (b) can contribute to the query top K results. The RESQ algorithm decide when and where to forward a query by modeling the problem as a linear program. The authors compared RESQ with two baselines: one which greedily forwards queries just to reduce energy cost, and one which greedily forwards queries just to maximize result quality. Simulations show that RESQ beats both the baseline in term of energy efficiency (46% energy cost saving are reported) while showing acceptable result quality and response times.

In a different line of research, Hidalgo et al. envision a collaboration between Internet Service Providers (ISPs) and web search engines to reduce network traffic and energy consumption [56]. The authors observe that many ISPs provide set-top-boxes (STBs) to their customers. STBs are actual computers that enable Internet connection and other services (like IPTV) into customer houses. These boxes are typically kept switched on, even when they are not performing demanding computations, and they can be remotely controlled by the ISP. Therefore, the authors propose to use these STBs to implement a distributed result cache for search engines. When an ISP's customer issue a query to a search engine, its results are cached into some STBs in a distributed fashion. Subsequent queries are then checked against this cache. If a query triggers a cache hit, results are served by the STBs. Otherwise, the query is processed by the actual search engine. In this way, ISPs could lower their network traffic towards search engines, then reducing their own operational costs. On the other hand, search engines can decrease their workloads, keeping fewer servers active and hence saving energy. Simulation results show that this caching mechanism could reduce by 40% the energy consumption of search engines.

Finally, it is worthy to mention the work of Hatzi, Cambazoglu, and Koutsopoulos on green web crawling, even if not related to query processing [55]. In their work, the authors observe that downloading a web page from a web server increases its energy consumption, as hardware resources are used to serve the requested page. As a consequence, the carbon footprint of the web server increases as well. However, the authors also observe that the carbon footprint of a web server depends on the type of energy which powers it. For example, a server is more likely to consume green energy at daytime, when solar power plants are functioning, and a crawler should exploit this fact to reduce the carbon footprint of its activities. Therefore, the authors introduce the problem of green web crawling, which aims at reducing the carbon footprint of a web crawler. The problem takes into account both the required freshness of downloaded pages and the carbon emissions of web servers. The authors propose various heuristics for scheduling the download of web pages. Experimental results show that the carbon footprint of a crawler can be considerably reduced without compromising the freshness of its document collection. Hatzi, Cambazoglu, and Koutsopoulos further extend their research in [54].

2.3 Positioning with respect to the state of the art

In this thesis, we address two main research questions: (a) is it possible to reduce the energy consumption of search servers?, and (b) is it possible to reduce the carbon footprint and energy expenditure of a multi-center search engine? In both cases, we want to improve the energy efficiency of search systems while minimizing any negative impact on its performance.

Regarding the first research topic, in this thesis we introduce a set of energy savings techniques which operate at the intra-server level, and which can be adopted at the inter-data center level as they are completely decentralized. In Chapter 4, we propose to delegate the control of the DVFS mechanisms from the operating system to the search application. This permits to exploit search servers' knowledge (e.g., utilization, load) to appropriately throttle the CPU frequency. We found that approaches we propose in Chapter 4 save up to $\sim$24% power with respect to a system which operates at maximum CPU frequency. When compared to systems that use more energy efficient configurations [17], we found that our approaches can still save at least 7% in power consumption. We believe that our techniques can be used in conjunction with energy efficient caching [78, 96], workload consolidation [42], and query forwarding mechanisms [13, 61, 101]. Approaches similar to ours are studied in [37, 71]. However, Lo et al. report several challenges in deploying their centralized solution on large clusters [71]. On the other hand, our approach is decentralized as it works at the search server level. In order to work, the technique from [37] requires to know the exact processing time of a query before its processing. Differently, our approaches do not rely on this assumption.

In Chapter 5 we further investigate our first research topic. We introduce the Predictive Energy Saving Online Scheduling (PESOS) algorithm, which exploits DVFS mechanisms to reduce the CPU energy consumption in a search node. Differently from the contributions of Chapter 4, PESOS takes into account also latency requirements, i.e., the search server is explicitly required to process queries by their deadlines. We experimentally show that PESOS outperforms OS frequency governors [103] and our best technique from Chapter 4. Moreover, PESOS does not interrupt query processing before its completion to meet latency requirement, unlike what happens in DES [37]. Additionally, PESOS estimates the query processing times using query efficiency predictors [75] while DES unrealistically assumes that the exact processing time of a query is known before its processing. We believe that PESOS can be used together with caching, workload consolidation, and query forwarding techniques [13, 42, 61, 78, 96, 101] to deploy more energy-efficient search architectures. On the contrary, the integration of PESOS with the approach proposed in [71] needs to be investigated, since both techniques require to control the CPU power management.

Finally, in Chapter 6 we focus on our second research topic. We propose a mathematical model to minimize the operational costs of multi-center web search engines by exploiting renewable energies whenever available at different locations. Using this model, we design a query forwarding algorithm which target the usage of green energy sources. Our work differs from [61, 101] as their solutions do not take into account the presence of renewable energies. Similar to our contribution, the work in [64] focuses on capping the carbon dioxide emissions of generic Internet services while our work take into account aspects specific to web search engines. In particular, we explicitly consider that search engines have strict latency constraints and query result quality can be degraded when these requirements cannot be met.

Chapter 3

Query Energy Consumption in Web Search Engines

3.1 Introduction

As seen in the Chapter 2, web search engines consume huge amounts of energy to process user queries, and energy costs have been explicitly considered in the design of caching [96] and query forwarding mechanisms [13, 61, 101] to reduce the energy expenditure of search engines. To the best of our knowledge, however, there is no study quantifying query energy consumption, i.e., the energy consumed by a search engine to process a query. Therefore, in this chapter we experimentally measure energy consumption on a per query basis. In particular, we focus on measuring the energy consumed by the CPU as it dominates the consumption of search server [10]. Measurements of the query energy consumption will be useful to motivate the following chapters of this thesis.

3.2 Experimental setup and analysis

To measure query energy consumption, we experiment using the Terrier IR platform [76] on a dedicated Ubuntu 14.04 server; Linux kernel version is 4.4.0-72-generic. The machine is equipped with 32GB RAM and an Intel i7-4770K multi-core processor. This CPU exposes 15 operating clock frequencies, ranging from 800 MHz to 3.5 GHz. The inverted index used in the experiments is obtained by indexing the ClueWeb09 (Cat. B) corpus [67]. The Porter stemmer is applied to every term and stopwords are removed. Document identifiers and term frequencies are compressed with Elias-Fano encoding [107]. Finally, the inverted index is kept in main memory.

We randomly sample 8,578 unique queries from the MSN 2006 query log [83]. These queries are processed sequentially by a single thread. In this way, only a CPU core is operative, while the operating system can set the remaining cores into an energy-saving, idle state. Query results are scored using the BM25 ranking function [95] to retrieve the top 1,000 documents. Results are nor stored neither sent over the network, to avoid energy demanding I/O interactions. Query statistics are reported in Table 3.1.

For each query, we measure its processing time and the energy consumed by the CPU. Energy consumption is measured from within Terrier, leveraging the Intel Running Average Power Limit (RAPL) interface via the jRAPL library [69]. Hackenberg et al. show the reliability of the RAPL interface in [52], and we confirm their findings in Appendix C.

Queries are processed using the MaxScore [105] and WAND [16] algorithms, to highlight possible differences in energy consumption. Also, we process queries using

Table 3.1: Distribution of queries across the various query lengths (number of terms).

	1	2	3	4	5	6+
#	2,180	2,999	2,019	880	343	157

Table 3.2: Minimum, mean, and maximum query processing times (in milliseconds).

Frequency	MaxScore			WAND		
	Min	Mean	Max	Min	Mean	Max
800 MHz	2	383	4504	2	490	6672
2.1 GHz	1	156	1769	1	195	2532
3.5 GHz	1	98	1039	1	120	1552

different core frequencies to quantify their impact on the query processing times and energy consumption. We use three frequencies: (1) the maximum (3.5 GHz), (2) the minimum (800 MHz), and (3) an intermediate one (2.1 GHz) among those available on our CPU.

As expected, Table 3.2 shows that processing queries with a high CPU frequency (3.5 GHz) reduces processing times. Indeed, we observe that the maximum query processing time is above 4 seconds when MaxScore is used and the CPU operates at 800 MHz. Instead, the maximum processing times reduces to about a second when the CPU frequency is 3.5 GHz.

However, high CPU frequencies entail also higher query energy consumption. As reported in Table 3.3, CPU energy consumption ranges from 0.004 to 5.842 Joules when queries are processed with MaxScore at 800 MHz, and from 0.008 to 10.180 Joules when the same queries are processed at 2.1 GHz. A peak of 16.847 Joules is consumed when the CPU operates at 3.5 GHz.

Moreover, we observe higher energy consumption when processing queries with WAND with respect to MaxScore. However, we explain this behavior due to the longer query processing times of WAND, as reported in Table 3.2.

Figure 3.1 shows query energy consumption and processing times. Each point in the figures represents a query, showing its processing time on the x-axis and its CPU energy consumption on the y-axis. Interestingly, from Figure 3.1 we can derive that the energy consumed for solving a query is linear in its processing time. For each operating frequency, in fact, we observe that the longer a query's processing takes to complete, the more energy is consumed by the CPU. Therefore, if the CPU has just one operating frequency, the most energy efficient way to process queries is to

Table 3.3: Minimum, mean, and maximum query energy consumption (in Joules).

Frequency	MaxScore			WAND		
	Min	Mean	Max	Min	Mean	Max
800 MHz	0.004	0.495	5.842	0.004	0.621	9.330
2.1 GHz	0.008	1.003	10.180	0.007	1.295	16.609
3.5 GHz	0.016	1.631	16.847	0.016	2.058	25.763

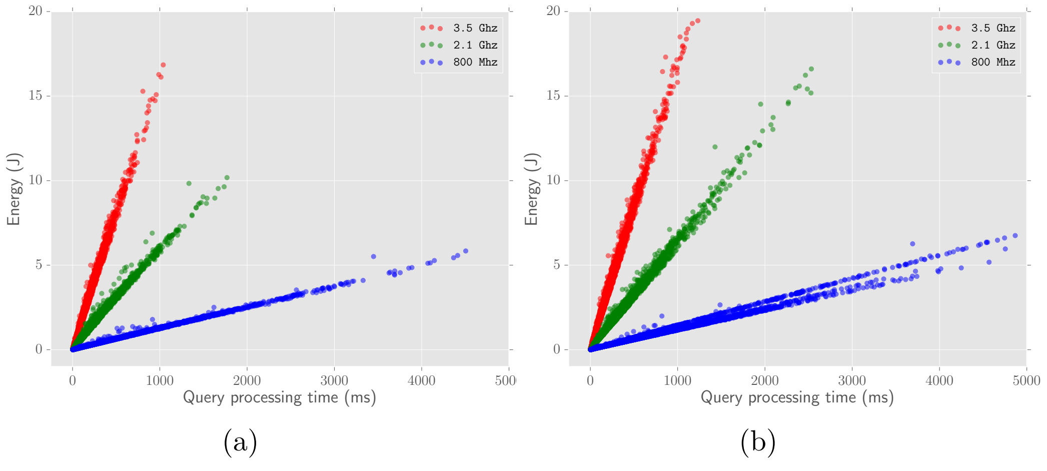


FIGURE 3.1: Query energy consumption and processing times with MaxScore (a) and WAND (b).

complete their processing as soon as possible. However, modern CPUs can operate at different frequencies with wide differences in query energy consumption. For example, Figure 3.1 shows that processing a query for 1 second at 3.5 GHz consumes more energy than processing a query for the same amount of time at 800 MHz.

This finding suggests that there are opportunities to exploit CPU frequency scaling to reduce the energy consumption of web search engines. In particular, search engine should process queries as fast as required to satisfy their users (e.g., within 500 ms since query arrival) [5]. However, queries should not be processed faster than necessary, i.e., low CPU frequencies can be used to reduce CPU energy consumption if they do not prolong query latencies above user expectations.

3.3 Discussion

In this chapter, we presented an experimental setting to measure the CPU energy consumption incurred by a search server to answer a single query. To be able to measure the energy consumption of a single query is important, as recent approaches are taking into direct account the search engine power consumption to achieve energy and money savings [13, 61, 96, 101].

The experiment results show that query energy consumption is linear in the query processing time. This indicates that efficient query processing is fundamental to tackle the economical and environmental challenges posed by web search engines. However, modern CPUs can operate at several clock frequencies. Higher frequencies produce short query processing times but they also entail high CPU energy consumption. On the contrary, low CPU frequencies reduce energy consumption but prolong processing times. We here advocate that search engine should exploit frequency scaling to process queries fast enough to keep users satisfied, but no faster than necessary to reduce energy consumption. We explore such hypothesis in Chapters 4 and 5.

Alternatively, a search engine's data centers can process queries as fast as possible, if these queries have been forwarded from other data centers. By using the maximum CPU frequency, the receiving data center can compensate the network delays incurred by query forwarding. At the same time, this allows to conveniently exploit query forwarding to process queries where electricity is cheap or provided by green sources, without incurring in latency violations. We explore this approach in Chapter 6

Chapter 4

Load-sensitive CPU Power Management for Web Search Engines

4.1 Introduction

In this chapter, we focus on the energy efficiency optimization of a search server within a web search engine's data center. As seen in Chapter 1, the power consumption of a typical server is dominated by its CPU. This is particularly true at low utilization levels, where CPUs consume a fixed amount of power without performing any demanding computations. For this reason, Barroso, Clidaras, and Hölzle report the data centers' need for energy-proportional computing, i.e., for hardware components with power consumption proportional to utilization [10].

As illustrated in Chapters 2 and 3, Dynamic Voltage and Frequency Scaling (DVFS) technologies sacrifices CPU performance for lower power consumption, by throttling processor's frequency. By doing so, CPUs operating at low frequencies absorb less power but have lower performance than CPUs working at higher frequencies. Operating system (OS) kernels can exploit DVFS to achieve energy savings, for instance by throttling the server CPU frequency accordingly to the processor utilization [17]. When the processor is under-utilized, a low CPU frequency is selected. Conversely, a high CPU frequency is picked when the processor is heavily utilized by the system. However, the OS misses domain-specific information on the search engine software and its interactions with the incoming user queries. We advocate that this information can be exploited to improve the energy proportionality of a search engine, as identified in [10].

In this chapter we propose search engine-specific frequency governors that can better adapt to varying query workloads by leveraging domain-specific information. While DVFS states are typically managed by the OS, their functionality can be (partially) controlled by application-level code [17]. We build upon this functionality to develop our proposed solution i.e., search engine-specific frequency governors which control the CPU frequency from within the search application. Indeed, knowledge of search server utilization and load facilitate a more refined control of the processor to achieve power savings.

The contributions introduced in this chapter are as follows: (1) we propose to exploit search servers' knowledge (e.g., utilization, load) to throttle the CPU frequency via search engine-specific frequency governors, and (2) we experimentally demonstrate that our solutions can achieve significant power savings without markedly damaging the query processing quality with respect to standard frequency governors.

The chapter is structured as follows. Section 4.2 formally describes the problem under study while Section 4.3 illustrates the proposed framework. Section 4.4 describes the setup of the experiments conducted to answer our research questions. Results are analyzed in Section 4.5 and conclusions follow in Section 4.6.

4.2 Problem statement

A search server can be modeled as a first-come first-served queue, where incoming queries wait to be processed upon arrival to the search engine. Queries are processed by a search application's thread, and we assume that each core of the server's CPU runs one thread. As soon as a processing thread is available, it picks the next query from the queue and starts processing it[1]. Queries arrive to the system with an arrival rate λ and are processed at a processing rate μ (both expressed in QPS, i.e., queries per second). The query arrival rate can vary over time, due to fluctuations in the query load [98]. The query processing rate may change as well, because of the DVFS mechanism: for lower CPU frequencies we expect lower power consumption but also lower μ values, as the CPU speed is reduced and query processing takes longer.

Since search engines users are impatient [97], we assume that search servers must process queries within a short time threshold τ since their arrival, e.g., 1 second [5]. However, since a query can spend some time waiting in the queue, processing threads may actually have less than τ seconds to solve certain queries. Additionally, execution times are variable and some queries may require more time than others to be processed [19]. If our system cannot complete a query processing within the time budget, the retrieval phase is terminated early and results computed so far are returned [33, 61]. This partial processing will likely have a negative impact on the effectiveness of the returned results; however, they can benefit from subsequent effectiveness-improving processing stages, e.g., machine-learned ranking [104]. Conversely, when a query exceeds τ seconds waiting in the queue, the server just drops it. In this case, the system returns no results.

As illustrated in Chapter 2, the power consumed by a server (measured in Watts) can be divided into two components: a static part which is continuously consumed to operate the hosting machine, and a dynamic (or operational) part which depends on the CPU usage to perform query processing activities. In this work, we propose to exploit DVFS technology to dynamically change the search server CPU frequency to reduce the operational power consumption. Indeed, the average operational power consumed by a server increases with the CPU frequency, as shown in Chapter 3. Of course, the operational power consumed by the server varies for different incoming workloads, since the machine utilization varies with the number of query arrivals [10].

Clearly, lower CPU frequencies consume less power but also increases the number of unanswered and partially processed queries, as lower frequencies decrease the query processing rate μ. Hence, our goal is to reduce the power consumed by a search server to process queries, while providing an acceptable query processing quality.

4.3 Proposed solution

To achieve our goal, we propose to move CPU power management from the OS directly to the search server application code. The Linux OS leverages DVFS to reduce power consumption. A CPU exposes a finite set of available operational frequencies

[1]We assume that queries are processed in disjunctive mode, i.e., documents must contain at least one query term to appear among the query results.

to the OS kernel, which exploits software modules called frequency governors [17] to dynamically select the current operational frequency. OS governors vary the processor frequency according to metrics like the CPU utilization, i.e., the fraction of time a processor is busy performing computations. For instance, the `conservative` governor steps up the processor frequency when the CPU utilization is above a tunable threshold α (e.g., 0.8). Conversely, the CPU frequency is scaled down if utilization is below a tunable threshold β (e.g., 0.2). In any case, the OS governors select the CPU frequency using kernel-level information. They do not use application-specific information from the search engine, which we argue can help to improve the system energy efficiency. For this reason, we propose to delegate frequency scaling decisions to application-level modules, by implementing governors inside the search engine. In the following, we will refer to `conservative` as `os-cons` for brevity and for distinguish it from our approaches.

We develop two search engine-specific frequency governors, based on search server knowledge: `se-cons` and `se-load`. The `se-cons` governor is inspired by the `os-cons` module, but it exploits the search server utilization instead of the raw CPU utilization. By using the query arrival and processing rates, the search server utilization ρ is computed as

$$\rho = \frac{\lambda}{k \cdot \mu} \tag{4.1}$$

where k is the number of threads processing queries [53]. The idea behind this governor is to maintain an acceptable search server utilization (e.g., 0.7 [53]) so that incoming queries can be easily processed without consuming too much power. Periodically, `se-cons` computes the search server utilization and adjusts the CPU frequency. Frequency throttling is performed if ρ is above (respectively below) the tunable threshold α (respectively β). If an adjustment is required, `se-cons` changes the processor frequency to obtain the desired utilization. The governor assumes that it will receive in the immediate future the same number of queries received during the last period. Using Equation 4.1, it computes the query processing rate necessary to obtain the target utilization. Finally, this governor selects the lowest frequency capable of producing such query processing rate, assuming processing rate directly proportional to CPU speed.

Our second governor, `se-load`, bases its frequency scaling decisions upon the number of queries N populating the search server, i.e., the queries currently queued or being processed. Given N, we define the search server load ℓ as:

$$\ell = \frac{N}{k} \tag{4.2}$$

Here, the principle is to reduce the query population in the server as fast as possible, when the search server load is too high (e.g. $\ell > 0.7$). Periodically, `se-load` observes the search server load and accordingly adjust the CPU frequency. If ℓ is greater than α, the processor is set to its maximum frequency. When ℓ is below β, the CPU frequency is stepped down from its current frequency to the next smaller one.

In the following section, we experiment in order to evaluate (a) how much power can be saved by using the search engine-specific frequency governors and (b) the corresponding impact on query processing quality.

4.4 Experimental setup

Experiments are conducted using the Terrier IR platform [76]. The platform is hosted on a dedicated Ubuntu 14.04 server; Linux kernel version is 3.13.0-45-generic. The machine is equipped with 32GB RAM and a 4-core Intel i7-4770K processor. The processor can run 8 threads by using hyperthreading, and it exposes 15 operational frequencies ranging from 800 MHz to 3.5 GHz. The ClueWeb09 (Cat. B) [67] document collection is indexed to represent the first tier of a web search engine. Stopwords are removed and the Porter stemmer is applied to all terms. The index stores document identifiers and term frequencies. The index is compressed with Elias-Fano encoding [107], and is kept in memory, shared among 8 query processing threads.

Queries are taken from the MSN 2006 query log [83] and are submitted in real time to our system, while halving their original interarrival time. Since we use the first day of the dataset, every experiment take 12 hours to run; the average query load in 11.28 QPS instead of the original 5.14, with a peak of 44 QPS instead of 28.

For each query, we use BM25 [95] to retrieve the top 1000 documents using WAND [16]. Upon arrival, queries are queued and have 1 second to be processed [2]. In fact, search engine users are very likely to perceive delays higher than 1 second [5]. Therefore, query processing is early terminated and partial results are returned if a second expires before processing completion. If a query spends all its time in the queue, the system drops the query. The query is unanswered and an empty result list is returned.

For each experiment, we measure (a) the % of unanswered queries (%UQ), (b) the mean recall, relatively to an ideal system which has infinite time to process every query (RR) and (c) the power consumed by the search server (P). In particular, power consumption is measured at the server power socket by using an Alciom PowerSpy2 wattmeter [3]. Measurements consider only the dynamic power consumption, i.e., we remove the power consumed by the server when idle (∼41.8 Watt). Power is measured every 30 milliseconds and the mean value is reported for each experiment.

Our baselines are given by standard Linux frequency governors. In particular, we compare our approach to two baselines: `performance` which processes every query at the maximum CPU frequency (we will refer to it as `perf` for brevity); and `os-cons` which adjusts the CPU frequency based on the processor utilization.

For `os-cons` frequency throttling decisions are taken every 0.08 milliseconds (the default value). Instead, our governors take decisions at every second, since we observe from the query log that the query arrival rate fluctuates every ∼1.2 seconds in average.

While `perf` is parameterless, other governors are tested under two different configurations. One has relaxed thresholds ($\alpha = 0.8$, $\beta = 0.2$ – as in the `os-cons` default setting), so that the search server is likely to maintain a certain CPU frequency for longer periods. The other configuration has tighter thresholds (selected as $\alpha = 0.8$, $\beta = 0.6$), so that the server will promptly react to changes in utilization or load.

We consider our search engine-specific governors successful if they show reduced power consumption (P) than the baselines, without marked degradation to query processing quality in terms of relative recall (RR) and % of unanswered queries (%UQ).

Table 4.1: Percentage of unanswered queries (%UQ), mean relative recall (RR) and mean consumed power (P, in Watt) for different frequency governors under various settings of α, β.

Governor	α	β	%UQ	RR	P
`perf`	-	-	0.342	**0.931**	41.765
`os-cons`	0.8	0.2	**0.283**	0.929	38.569
	0.8	0.6	0.295	0.927	35.381
`se-cons`	0.8	0.2	0.352	0.911	36.016
	0.8	0.6	0.315	0.900	**31.727**
`se-load`	0.8	0.2	0.312	0.913	35.455
	0.8	0.6	0.292	0.912	32.888

4.5 Results

Experiments results are reported in Table 4.1. We observe that the `perf` baseline consumes the highest operational power (∼42 Watt) and causes the search server to drop more than 0.3% of the incoming queries. However, under this configuration the system shows the best relative recall (0.931). Relative recall values are statistically significant according to paired t-tests ($p < 0.01$).

The search server using the `os-cons` governor shows reduced power consumption with respect to a server equipped with the `perf` governor. Indeed, `os-cons` can consume from ∼8% to ∼15% less power than a governor which always maintains the CPU at the maximum frequency. Also, `os-cons` drops fewer queries but provides a slightly worse relative recall.

Our first search engine-specific governor, `se-cons`, leads to reduced power consumption if compared to `os-cons` runs. In fact, our governor saves more than 6% in power consumption when relaxed thresholds are set ($\alpha = 0.8$, $\beta = 0.2$); and more than 10% using tight thresholds ($\alpha = 0.8$, $\beta = 0.6$). These power savings come at the price of small degradation in query processing quality: the percentage of unanswered queries increases by ∼6% while the relative recall decreases by almost 2%, if we compare `se-cons` to `os-cons` with tight thresholds. Under the relaxed threshold, `se-cons` drops ∼24% more queries than `os-cons`, while its relative recall diminishes of ∼3% in comparison. When compared to `perf`, `se-cons` can help save from ∼14% to ∼24% in power consumption. Relative recall decreases by slightly more than 2% when using relaxed thresholds, and by almost 3% with tight ones. The percentage of unanswered queries increases of ∼3% with respect to `perf` when `se-cons` uses relaxed thresholds. However, dropped queries decrease by ∼8% when tight thresholds are set.

Our second governor, namely `se-load`, obtains power savings similar to the `se-cons` governor, but with a better query processing quality. Indeed, `se-cons` saves more than 8% in power consumption when compared to `os-cons` with relaxed thresholds. However, the relative recall detriment is less than 2% and the unanswered queries increment by just ∼10%. When the governors are configured with tight thresholds, `se-load` saves 7% in power consumption w.r.t. `os-cons`. At the same time, relative recall is damaged for less than 2% and no additional queries are dropped. When compared to `perf`, `se-load` saves from ∼15% to ∼21% in power consumption. Relative recall is damaged by ∼2% under both threshold configurations.

[2]Regarding the queries used in our experiments, approximatively 7% of them take at least one second to be processed at maximum CPU speed.

Instead, the percentage of unanswered queries benefits from our governor. Under relaxed thresholds, `se-load` drops ∼9% fewer queries than `perf` and ∼15% fewer queries remain unanswered by using tight thresholds.

Overall, experiments confirm that our approach is successful, as the search engine-specific governors show reduced power consumption than the two baselines. In particular, `se-cons` provides the highest power saving. Relative recall (RR) and percentage of unanswered queries (%UQ) are not markedly damaged, especially when using `se-load`.

4.6 Discussion

In this chapter, we advocated that search engines infrastructures can save power at search server level, by leveraging knowledge on the server querying operations. We developed two search engine-specific frequency governors, `se-cons` and `se-load`, which perform processor frequency throttling according to the search server utilization and load. By extensive experimentation, we evaluated the benefits and drawbacks of our approaches, compared to standard OS-level frequency governors. We found that `se-cons` can help save up to ∼24% power with respect to a system which operates at maximum CPU frequency to promote query processing quality. Indeed, `se-cons` damages by just ∼3% both relative recall and percentage of unanswered queries. When compared to systems that use more energy efficient configurations, we found that our governors can still save at least 7% in power consumption. This gain costs only a limited detriment in relative recall (less than 2%) when `se-load` is used. Greater power savings can be achieved by accepting more substantial degradation in query processing quality.

Such power savings are important at data center-level too. Indeed, low CPU frequencies reduce heat output and, as a consequence, thermal cooling expenditure [10]. Moreover, `se-cons` and `se-load` are completely decentralized and can deployed in a replicated search architecture. We plan to study the efficacy of our contributions at the inter-data center level in a future work.

Chapter 5

Energy-efficient Query Processing in Web Search Engines

5.1 Introduction

As showed in Chapters 3 and 4, CPU frequency scaling can be leveraged to reduce the energy consumption of search servers. Indeed, low CPU frequencies entails low power consumption but they also prolong query processing times. However, users can hardly notice response times that are faster than their expectations [5]. Therefore, to reduce energy consumption, web search engines should answer queries no faster than user expectations. As discussed in Chapter 2, several power management policies leverage DVFS technologies to scale the frequency of CPU cores accordingly to their utilization [17, 103]. However, core utilization-based policies have no mean to impose a required latency on a search server. As a result, the server can consume more energy than necessary in serving queries faster than required, with no benefit for the users.

In this chapter we propose the Predictive Energy Saving Online Scheduling algorithm (PESOS), which considers latency requirement of queries as an explicit parameter. Via the DVFS technology, PESOS selects the most appropriate CPU frequency to process a query on a per-core basis, so that the CPU energy consumption is reduced while respecting a required latency. The algorithm bases its decision on *query efficiency predictors* rather than core utilization.

Query efficiency predictors (QEPs) are techniques to estimate the processing time of a query before its processing. To know in advance the execution time of queries permits to improve the performance of a search engine. Most QEPs exploit the characteristics of the query and the inverted index to precompute features to be exploited to estimate the query processing times. For instance, Macdonald et al. [75] propose to use term-based features (e.g., the inverse document frequency of the term, its maximum relevance score among others) to predict the execution time of a query. They exploit their QEPs to implement online algorithms to schedule queries across processing node, in order to reduce the average query waiting and completion times. The work in [59, 62] addresses the problem to whether parallelize or not the processing of a query. In fact, parallel processing can reduce the execution time of long-running queries but provides limited benefits when dealing with short-running ones. Both the works propose QEPs to detect long-running queries. The processing of the query is parallelized only if their QEPs detect the query as a long-running one. Rather than combining term-based features, Wu et al. [112] propose to analytically model the query processing stages and to use such model to predict the execution time of queries. However, to the best of our knowledge, query efficiency predictor have not been considered for reducing the energy consumption of search servers.

In this chapter, we build upon the approach described in [75] and propose two novel query efficiency predictor techniques: one to estimate the number of postings that must be scored to process a query, and one to estimate the response time of a query under a particular core frequency given the number of postings to score. PESOS exploits these two predictors to determine which is the lowest possible core frequency that can be used to process a query, so that the CPU energy consumption is reduced while satisfying the required latency. As predictors can be inaccurate, in this chapter we also propose and investigate a way to compensate prediction errors using the root mean square error of the predictors.

We experimentally evaluate PESOS upon the TREC ClueWeb09 [67] corpus and the query stream from the MSN2006 query log [83]. We compare the performance of our approach with those of three baselines: (1) `performance` (`perf`, for brevity) [103], which always uses the maximum CPU core frequency, (2) `powersave` (here referred to as `power`[1]) [103], which throttles CPU core frequencies according to the core utilizations, and (3) `se-cons`, which we introduced in Chapter 4 and which performs frequency throttling according to the search server utilization. PESOS, with predictors correction, is able to meet the latency requirements while reducing the CPU energy consumption from ∼24% up to ∼44% with respect to `perf` and up to ∼20% with respect to `se-cons`, which however incurs in uncontrollable latency violations. Moreover, the experiments show that energy consumption can be further reduced by PESOS when prediction correction is not used, but with higher latencies.

The rest of the chapter is structured as follows: Section 5.2 formulates the problem of minimizing the energy consumption of a search server while maximizing the number of queries which meet their deadlines. Section 5.3 illustrates our proposed solution to the problem, describes our query efficiency predictors, and the PESOS algorithm. Section 5.4 illustrates our experimental setup while Section 5.5 analyzes the obtained results. Finally, the chapter concludes in Section 5.6.

5.2 Problem statement

In the following, we introduce the operative scenario of a query processing node (Sec. 5.2.1), we formalize the general minimum-energy scheduling problem and we shortly present the state-of-the-art algorithm to solve it offline (Sec. 5.2.2), and we discuss the issues of this offline algorithm in our scenario (Sec. 5.2.3).

5.2.1 Operative scenario

A *query processing node* is a physical server composed by several multi-core processors/CPUs with a shared memory which holds the inverted index. As discussed in Chapter 2, the inverted index can be partitioned into *shards* and distributed across multiple query processing nodes. In this chapter, we focus on reducing the CPU energy consumption of single query processing node, independently of the adopted partition strategy. In the following, we assume that each query processing node holds an identical *replica* of the inverted index [43].

A *query server* process is executed on top of each of the CPU core of the processing node (see Figure 5.1). All query servers access a shared inverted index held

[1]`power` is a frequency governor developed by Intel which replaced the `conservative` governor in the latest Ubuntu distributions.

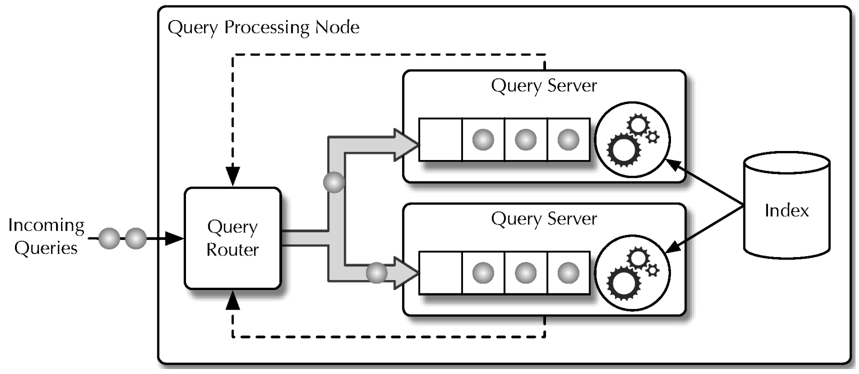


FIGURE 5.1: The architecture of a query processing node.

in main memory to process queries. Each query server manages a queue, where the incoming queries are stored. The first query in the queue is processed as soon as the corresponding CPU core is idle. The queued queries are processed following the *first-come first-served* policy. The number of queries in a query server's queue represents the server load. Queries arrive to the processing node as a stream $S = \{q_1, \dots, q_n\}$. When a query reaches the processing node, it is dispatched to a query server by a *query router*. The query router dispatches an incoming query to the least loaded query server, i.e., to the server with the smallest number of enqueued queries. Alternatively, the query processing node could have a single query queue and dispatch queries from the queue to idle query servers. In this chapter, we use a queue for each query servers since a single queue will not permit taking local decisions about the CPU core frequency to use for the relative query server. A similar queue-per-core architecture is assumed in [2], to schedule jobs across CPU cores to minimize the CPU energy consumption, and in [75] to schedule queries across different query servers.

A query $q_i \in S$ is characterized by its arrival time a_i, when it "enters" the processing node at the query router, and its completion time $c_i > a_i$, when it "leaves" the processing node after being processed by a query server. The query processing node is required to process queries with a latency of τ ms to meet user expectations (e.g., 500 ms [5]). Therefore, we impose that each query q_i must be processed within τ time units from its arrival time, i.e., it has an absolute deadline $d_i = a_i + \tau$. If we assume negligible the time required by the query router to dispatch the query, the completion time c_i of q_i is the sum of its arrival time, the time the query spent in the queue and its processing time. A query misses its deadline, i.e., $c_i > d_i$, if it spends more than τ time units in queue and being processed. In fact, a query may have less than τ time units to be processed. At time t, the *time budget* $b_i(t)$ of query q_i indicates how much time remains before q_i misses its deadline. $b_i(t)$ is the difference between its deadline and the time it is spending in the queue, i.e. $b_i(t) = d_i - (t - a_i)$. When a query exceeds its time budget, the query processing node has two possible choices: 1) to early terminate the query, returning an incomplete list of results, or 2) to finish processing the query, delaying the processing of other request, but returning a complete list of results. Differently from Chapter 4, we here focus on the second option which does not degrade the quality of the search results. We do not consider here the time necessary to send the results to the users, as it involves network latencies which do not depend on the search engine.

As seen in Chapter 3, a query server can process queries at different speeds, depending to the CPU core operational frequency. To reduce deadline violations, CPUs cores can operate at their maximum processing frequency. In fact, high frequencies lead to faster computations at the price of high power consumption. Conversely, lower frequencies mean slower computations, with lower power consumption.

Since the number of queries received by a query processing node along a day varies, we envision the possibility to dynamically change the CPU core frequencies of query servers to match the number of queries received per time unit. Our goal is to maximize the number of queries that are processed within their deadline, in order to obtain a latency close to τ ms. At the same time, we want to minimize the energy consumption of the processing node. In other words, for each query q_i we need to select the most appropriate frequency $f \in F$ for the CPU core associated to the server processing q_i.

5.2.2 The minimum-energy scheduling problem

Consider the following scenario, where a single-core CPU must execute a set $J = \{J_1, \ldots, J_n\}$ of generic computing jobs rather than queries. Jobs must be executed over a time interval $[t_0, t_1]$. Each job J_i has an arrival time a_i and an arbitrary deadline d_i which are known a priori. Moreover, each job J_i has a processing volume v_i, i.e., how much work it requires from the CPU, and jobs can be preempted. The CPU can operate at *any* processing speed $s \in \mathbb{R}^+$ (in time units per unit of work) and its power consumption is a convex function of the processing speed, e.g., $P(s) = s^\gamma$ with $\gamma > 1$ [99].

Jobs in J must be scheduled on the CPU. A *schedule* is a pair of functions $S = (\psi, \phi)$ denoting, respectively, the processing speed and the job in execution, both at time t. A schedule is *feasible* if each job in J is completed within its deadline. The *minimum-energy scheduling problem* (MESP) aims at finding a feasible schedule such that the total energy consumption is minimized, i.e.,

$$\underset{S=(\psi,\phi)}{\arg\min}\, E(S) = \int_{t_0}^{t_1} P(\psi(t))dt \tag{5.1}$$

The MESP is similar to an offline version of our problem, where jobs, corresponding to queries, are preemptable, and processor speeds can assume any positive value.

The YDS algorithm [114] solves the MESP in polynomial time. Consider an interval $I = [z, z'] \subseteq [t_0, t_1]$ and the set of jobs in that interval $J_I = \{J_i \in J : [a_i, d_i] \subseteq I\}$. The *intensity* $g(I)$ of interval I is the ratio between the amount of work required by the jobs in J_I and the length of the interval

$$g(I) = \frac{1}{z - z'} \sum_{J_i \in J_I} v_i \tag{5.2}$$

A feasible schedule must use a processing speed $s \geq g(I)$ during the interval I, or jobs will not meet their deadlines if $s < g(I)$. Moreover, $P(g(I))$ is the lowest possible power consumption on the interval I, since P is a convex function.

Algorithm 1 illustrates the YDS algorithm, that optimally solves the MESP in $O(n^3)$ [9, 114]. YDS works by analyzing each possible time interval I included in $[t_0, t_1]$. Then, it finds the *critical interval* I^* that maximizes $g(I)$. YDS schedules the jobs in J_{I^*} using the *earliest deadline first* (EDF) policy [1] and processing speed $g(I^*)$. Then, if not preempted, the jobs in J_{I^*} will terminate in $r_i = v_i \cdot g(I^*)$ time units since the beginning of their execution. Jobs in J_{I^*} are then removed from J. The interval I^* as well is removed from $[t_0, t_1]$, i.e., it cannot be used to schedule jobs other than those in J_{I^*}. For this reason, YDS updates the arrival times and deadlines of the remaining jobs to be outside I^*. Finally, YDS repeatedly finds a new critical

Algorithm 1: The YDS algorithm

```
Data: A set of jobs $J = \{j_1, \dots, j_n\}$ to schedule in $[t_0, t_1]$
Result: A feasible schedule $S$ for $J$ minimizing $E(S)$
OYDS($J$):
1     $\psi \leftarrow \{\}$
2     $\phi \leftarrow \{\}$
3     while $J \neq \{\}$ do
4         Identify $I^* = [z, z']$ and compute $g(I^*)$
5         Set processor speed to $g(I^*)$ for jobs in $J_{I^*}$ in $\psi$
6         Schedule jobs in $J_{I^*}$ according to EDF in $\phi$
7         Remove $I^*$ from $[t_0, t_1]$
8         Remove $J_{I^*}$ from $J$
9         foreach $J_i \in J$ do
10            if $a_i \in I^*$ then
11                $a_i \leftarrow z'$                    // Update arrival times
12            if $d_i \in I^*$ then
13                $d_i \leftarrow z$                     // Update deadlines
14    return $S = (\psi, \phi)$
```

interval for the remaining jobs, until all jobs are eventually scheduled. Note that the MESP always admit a feasible schedule, since arbitrary large amounts of work can be performed in infinitesimal time when $s \to \infty$.

Figure 5.2 shows an example for YDS. Input jobs are illustrated in the upper part of the picture. The left end of a box indicates the arrival time of the job, while the right end indicates its deadline. Processing volumes for the jobs are reported inside the relative boxes. The bottom part of the picture illustrates the optimal solution provided by YDS. The picture shows the order in which the jobs are scheduled, their start and end time, and the processing speeds s used for each job. Note that J_3 is executed over two different time intervals, as it is preempted to schedule J_4 and J_5, which have a higher joint intensity.

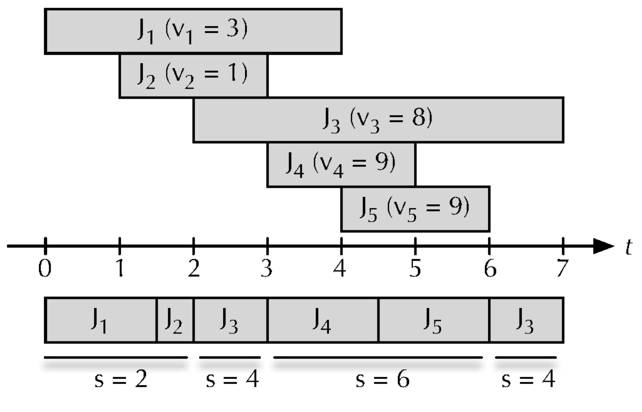


FIGURE 5.2: An example of YDS scheduling: (top) input jobs, (bottom) resulting optimal schedule with CPU speeds s.

5.2.3 Issues with YDS

YDS finds an optimal solution for the MESP, but poses various issues that make difficult to use it in a search engine to reduce its energy consumption:

1. YDS is an offline algorithm to schedule generic computing jobs and cannot be used to schedule online queries. In fact, YDS input is the set of jobs to be scheduled in a interval, with their arrival times and deadlines, that must be known a priori. In contrast, query arrival times are not known until query arrives. Moreover, YDS relies on EDF, which contemplates job preemption. Context switch and cache flushing cause time overheads with non-negligible impacts on the query processing time. Therefore, preemption is unacceptable for search engines.

2. YDS requires to know the processing volumes of jobs in advance. Conversely, we do not know how much work a query will require before its completion.

3. YDS schedules job using processing speeds (defined as units of work per time unit). The speed value is continuous and unbounded (i.e., the speed can be indefinitely large). However, the frequencies available to CPU cores are generally discrete and bounded.

For such reasons, in the following Section we modify YDS in order to exploit it in a search engine.

5.3 Problem solution

YDS has several issues that make unfeasible to use it in a search engine. In the following, we discuss:

1. a heuristic based on YDS which works in online scenarios without job preemption (Sec. 5.3.1),

2. a methodology to estimate the processing volume of a query (Sec. 5.3.2),

3. an algorithm to translate processing speeds into CPU core frequencies (Sec. 5.3.3).

Eventually, we introduce and discuss our approach to select the most appropriate CPU core frequency to process a query in a search engine (Sec. 5.3.4).

5.3.1 On-line scheduling without preemption

Online YDS[2] (OYDS) is an heuristic for the online version of the MESP, proposed in [114]. In an online scenario, we are not given a set of jobs over a fixed time interval, but the set of jobs that must be processed by the CPU changes over time. Every time $\hat{t}$ a new job arrives, OYDS considers the newly arrived job and all the jobs still to be (completely) processed, and computes an optimal solution using YDS for this set of jobs, assuming that all such jobs have the same arrival time $\hat{t}$. As YDS, OYDS guarantees that each job will be terminated by its deadline. In fact, it can schedule any processing volume by simply using an arbitrarily large processing speed s. On the other hand, its energy consumption can be sub-optimal.

[2]In the original paper, OYDS is called Optimal Available (OA). In this chapter, we will use OYDS for the sake of clarity.

While OYDS is a heuristic for the online version of the MESP, it still schedules jobs using the EDF policy which contemplates job preemption. However, in our operative scenario we deal with queries rather than generic computing jobs. Preemption is unacceptable for search engines and a query cannot be preempted once its processing has started. Since all queries must be processed within the same relative deadline τ, for any two queries q_h and q_k, such that $a_k > a_h$, we have $d_k > d_h$, i.e., later queries have later deadlines. As a consequence, EDF will always schedule firstly the earliest query, without any preemption. This means that, under these conditions, EDF coincides with the *first-in first-out* (FIFO) scheduling policy. We will use OYDS as a base for build our frequency selection algorithm, described in Section 5.3.4. In the remaining of this work, then, we will stop discussing about generic computing jobs but we will focus on the processing of search engine queries.

5.3.2 Predicting processing volumes

The OYDS heuristic must know the processing volumes of the queries to schedule. For this purpose, we propose to use the number of scored posting during the processing of query. Indeed, for queries with the same number of terms, the number of scored postings correlates with their processing times [75]. If exhaustive processing is performed, it is possible to know a priori the number of scored postings, which is equal to the sum of the posting lists lengths of the query terms. However, when dynamic pruning is applied we do not know in advance how many postings will be scored, since portions of the posting lists could be skipped. Then, we need a way to predict the number of scored posting for a query.

We use the query efficiency predictors (QEPs) described in [75] but we modify them to predict the number of scored postings for a query. This means that we learn a set Π of linear functions $\pi_x(q)$ that, given a query q with x query terms, estimate the number of scored postings.

We note that OYDS requires exact query processing volumes. If the reported processing volumes are less than the actual ones, the algorithm does not guarantee that all the queries deadlines will be meet. QEPs are not precise, but they give only an estimate on the number of scored postings. For this reason, we add an offline validation phase after the QEPs training. During the validation, we use the regressors in Π to predict the number of scored posting for a validation set of pre-processed queries. Then, we record the root mean squared error (RMSE) for the predictions. In the online query processing, we use the RMSE ρ_x of predictor π_x to compensate its errors, by adding ρ_x to the predicted number of scored postings. In other words, our modified QEPs $\widetilde{\pi}_x(q)$ will be

$$\widetilde{\pi}_x(q) = \pi_x(q) + \rho_x. \tag{5.3}$$

In this way, we will likely over-estimate the processing volume of some queries, requiring higher processing speeds at the cost of higher energy consumption. However, we will miss less deadlines, as we reduce the number of queries for which we predict fewer scored postings lower than the actual ones.

5.3.3 Translating processing speeds into CPU frequencies

CPU cores can operate at frequencies $f \in F$, where F is a discrete set of available frequencies (measured in Hz). Nevertheless, OYDS assigns processing speeds (seconds per unit of work) to queries. Therefore, we need to map processing speeds to CPU core frequencies. To do so, for each frequency f we train a single-variable linear predictor $\sigma_x^f(q)$, which forecasts the processing time of a query q composed by x terms at frequency f through the estimated number of its scored postings:

$$\sigma_x^f(q) = \alpha_x^f \widetilde{\pi}_x(q) + \beta_x^f, \tag{5.4}$$

where α_x^f and β_x^f are the coefficients learned by the regressors. Thus, we learn offline a new set Σ of single-variable linear regressors σ_x^f, one for each frequency f. Once again, we add a validation phase after the training to build Σ, similarly to approach described in Section 5.3.2. We compensate a predictor error adding its RMSE (ρ_x^f) computed over the validation queries to the actual prediction, i.e.,

$$\widetilde{\sigma}_x^f(q) = \sigma_x^f(q) + \rho_x^f. \tag{5.5}$$

We can use Σ to translate processing speeds to CPU core frequencies, as shown in Algorithm 2. When a query q_i is associated to a processing speed s by OYDS, we compute its required processing time r_i by multiplying the predicted number of scored postings $\widetilde{\pi}_x(q_i)$ by s. Then, we check each regressor $\widetilde{\sigma}_x^f(q_i)$ in Π' in ascending order of frequency f. If the expected query processing time at frequency f is less than r_i, we use frequency f to process q_i. If we are not able to find a suitable frequency f, we use the maximum available frequency.

As shown in Algorithm 2, a suitable frequency f among the frequencies of the CPU cores for a query q_i does not always exist. For example, this happens when the query server is overloaded with queries to process. However, we can ignore this scenario by assuming that a query processing node has a computing capacity that, at maximum frequency, is sufficient to process its peak query volume. Moreover, a suitable frequency for a query q_i cannot be found if, at time t, q_i requires a processing time that is greater than its time budget $b_i(t)$. In such cases, we use the maximum CPU core frequency to minimize that query processing time.

5.3.4 Frequency selection algorithm for search engines

In this section, we describe PESOS (Predictive Energy Saving Online Scheduling). PESOS is an algorithm to select the most appropriate frequency to process a query in a search engine. Our algorithm is based on OYDS, but exploits predictors which can be inaccurate. Due to wrong predictions (see Sec. 5.3.2 and Sec. 5.3.3), some queries will miss their deadline no matter the selected CPU core frequency. Yet, this can happen because either queries have low time budgets or they require too much

Algorithm 2: The CPU core frequency selection algorithm

```
Data: A query $q_i$ composed by $x$ terms, and the processing speed $s$ assigned
      by OYDS to $q_i$
Result: The core frequency $f$ to use to process $q_i$
SelectFrequency($q_i$, $s$):
1   $r_i \leftarrow \widetilde{\pi}_x(q_i) \cdot s$
2   foreach regressor $\widetilde{\sigma}_x^f$ in $\Sigma$, in ascending order of $f$ do
3       $r_i^f \leftarrow \widetilde{\sigma}_x^f(q_i)$
4       if $r_i^f \leq r_i$ then
5           return $f$
6   return $\max_{f \in F}\{f\}$
```

Algorithm 3: The algorithm to compute the shared tardiness of a query queue

```
Data: The query queue $Q$ and the current time $t$
Result: The shared tardiness quantity $H(Q)$
ComputeSharedTardiness($Q$, $t$):
1   $T \leftarrow 0$                                        // Total tardiness
2   $n \leftarrow 0$                                        // On time queries
3   $\bar{f} \leftarrow \max_{f \in F}\{f\}$                // Maximum frequency
4   foreach query $q_i$ in $Q$ do
5       $b_i \leftarrow \tau - (t - a_i)$                   // Remaining budget
6       $r_i^{\bar{f}} \leftarrow \widetilde{\sigma}_x^{\bar{f}}(q_i)$   // Max processing speed
7       if $r_i^{\bar{f}} > b_i$ then
8           $T \leftarrow T + (r_i^{\bar{f}} - b_i)$        // Late query
9       else
10          $n \leftarrow n + 1$                            // On time query
11  return $T/n$
```

processing time. We call these *late* queries. Conversely, we call *on time* queries those that will be completely processed by their deadline.

Given a query q_i with deadline d_i and completion time c_i, we define its *tardiness* as $T_i = \max\{0, d_i - c_i\}$. As such, an on time query will have 0 tardiness, while a late query will have a tardiness given by the amount of time a query requires to be completed exceeding its deadline. While missing a query deadline is always undesirable, low tardiness values are still better than higher ones. Therefore, we aim at minimizing the tardiness of late queries, by reducing the time budget of on time queries. Given a queue of queries Q sorted by arrival time, we compute the total tardiness of the late queries in Q when all queries are processed at maximum frequency. Then we compute the *shared tardiness* $H(Q)$ of the on time queries in Q by dividing the total tardiness by the number of on time queries in Q, and we reduce the on time queries' deadlines by $H(Q)$. Hence, on time queries are required to finish their processing earlier, but this will leave more time to late queries and reduce their actual tardiness. Algorithm 3 recaps the steps to compute the shared tardiness $H(Q)$.

Algorithm 4 describes how PESOS sets the most appropriate core frequency to process a query. The algorithm works as follows. Assume q_1 is the first query in the query queue Q of a query server. At time t, query q_1 begins being processed. Initially, we check if q_1 is going to meet its own deadline. If the query is late, we set the core at its maximum frequency. Otherwise, we compute the shared tardiness $H(Q)$ of the queued queries and we change the deadlines of all the queries in Q accordingly, i.e., for all q_i in Q, we set $\widetilde{d_i} = d_i - H(Q)$. In doing so, we should just reduce the time budgets of the on time queries to leave more time to late queries. In fact, reducing the time budget of late queries has no effect since late queries will be in any case processed at maximum core frequency. Nevertheless, we reduce all the time budget by $H(Q)$ such that, for each couple of queries $q_j, q_k \in Q$, if $d_j \geq d_k$ then $\widetilde{d_j} \geq \widetilde{d_k}$. This property ensures that queries will be processed following the FIFO policy, avoiding preemption (see Sec. 5.3.1). Then, we check if the query q_1 is going to miss its *modified* deadline. In such case, we set the core at maximum frequency. On the contrary, we eventually run the OYDS algorithm to select which core frequency to use. Note that we need to compute just the core frequency for the query q_1. Then, we do not need to analyze

Algorithm 4: The PESOS algorithm for setting the most appropriate CPU core frequency to process a query

```
Data: The query queue $Q$ and the current time $t$
Result: The CPU core frequency to use for processing the first query in $Q$
PESOS($Q$, $t$):
1   $\bar{f} \leftarrow \max_{f \in F}\{f\}$                          // Maximum frequency
2   $q_1 \leftarrow Q$.head()                                          // First query
3   if $d_1 < t$ then
4       return $\bar{f}$
5   $H(Q) \leftarrow$ ComputeSharedTardiness($Q, t$)
6   if $d_1 - H(Q) < t$ then
7       return $\bar{f}$
8   $g(I^*) \leftarrow 0$                                              // Maximum intensity
9   foreach query $q_i$ in $Q$ do
10      if $d_i - H(Q) < t$ then
11          return $\bar{f}$
12      $Q_I = \{q_j \in Q : d_j \leq d_i - H(Q)\}$
13      $V \leftarrow \sum_{q \in Q_I} \widetilde{\pi}_x(q)$                 // Volume
14      $g(I) \leftarrow V/(d_i - H(Q) - t)$                              // Intensity
15      if $g(I) > g(I^*)$ then
16          $g(I^*) = g(I)$
17  return SelectFrequency($q_1, g(I^*)$)
```

each time interval in the query queue Q. Instead, we will check only the time intervals $[t, \widetilde{d_i}] = [t, d_i - H(Q)]$ for all queries $q_i \in Q$. If a query in the queue is likely to miss its deadline, we use the maximum core frequency to process q_1 at maximum speed. Otherwise, once we have identified the critical interval I^* (see Section 5.2.2) and its intensity $g(I^*)$, we select the most appropriate core frequency to process the first query q_1 by using Algorithm 2.

PESOS is executed whenever a query server starts processing a new query. When the query processing is completed, the query is removed from the query queue Q. Also, PESOS is executed at each new query arrival, to take into account the increased workload in the query queue and to adjust the core frequency for the query which is currently being executed. PESOS runs in linear time. In fact, it computes the shared tardiness using Algorithm 3, which just need to traverse the query queue. Then, the algorithm checks each interval $[t, \widetilde{d_i}]$ for all $q_i \in Q$, i.e., it analyzes $|Q|$ intervals. Eventually, it translates a processing speed into a CPU core frequency using Algorithm 2. Algorithm 2 needs to analyze at most $|F|$ CPU frequencies. In conclusion, the computational complexity of PESOS is $O(|Q| + |F|)$.

5.4 Experimental setup

In this section, we firstly describe the experimental setup for the training and validation of our predictors (Sec. 5.4.1, Sec. 5.4.2). Then, we illustrate the experimental setup we adopt to measure the CPU energy consumption and the latency of a query processing node using our approach (Sec. 5.4.3). All the experiments are conducted using the Terrier search engine [76]. The platform is hosted on a dedicated server with 32 GB RAM. The operating system is Ubuntu, with Linux kernel version 3.13.0-79-generic. The machine is equipped with an Intel i7-4770K CPU, a member of the Haswell product family. The CPU has 4 physical cores which expose 15 operational frequencies $F = \{0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.1, 2.3, 2.5, 2.7, 2.9, 3.1, 3.3, 3.5\}$ GHz. The inverted index used in the experiments is obtained by indexing the ClueWeb09 (Cat. B) document collection which contains more than 50 millions of web pages [67]. On each document, we remove stopwords and apply the Porter stemmer to all of its terms. The inverted index stores document identifiers and terms frequencies and it is kept in main memory, compressed with Elias-Fano encoding [107]. For the queries, we use the MSN 2006 query log [67].

In our experiments, we process queries using two dynamic pruning retrieval strategies: 1) MaxScore [105], and 2) WAND dynamic pruning [16]. For each query, we retrieve the top 1,000 documents according to the BM25 ranking function. The node operates with 4 query servers, i.e., processing threads, which are pinned to different CPU physical cores and share the same inverted index.

5.4.1 Training processing volume predictors

In this section, we adapt the *query efficiency predictors* (QEPs) introduced in [75] to originally predict the response times of a query. Instead, we modify these predictors to estimate the number of scored postings for a query. We divide queries into six query classes according to their number of terms, i.e., the first class includes queries with one term, while the last class includes queries with six or more terms.

To train and validate our predictors, we extract a number of unique queries from the MSN 2006 query log. We use unique queries to avoid any caching mechanism from the operating system that could distort our measurements. For each query class, we extract 10,000 unique queries from the MSN 2006 query log, generating a query set of 60,000 unique queries.

Before training the modified QEPs, we process each single term in the query set as detailed in [75]. We treat single terms as queries of length one. During the processing, we record the ranking scores obtained by all the documents relative to the terms. By doing this, for each query term we obtain the following set of 13 term-based features:

- **Arithmetic, geometric, and harmonic mean score:** Means of the ranking function (e.g., BM25) scores from the postings.
- **Max score:** The maximum ranking function score in the posting list.
- **Score variance:** The variance of the ranking function score in the posting list.
- **Number of postings:** The posting list length.
- **Number of maxima:** The number of postings for which the ranking function score is a local maximum w.r.t. adjacent postings in the posting list.
- **Number of maxima greater than avg. score:** The number of postings for which the ranking function score is a local maximum greater than the average score.
- **Number of postings with max score:** The number of postings for which the ranking function score is equal to the max score.
- **Number of postings within 5% of max score:** The number of postings whose ranking function score is at most 5% less than the max score.

- **Number of postings within 5% of threshold:** The number of postings whose ranking function score is at most 5% less than the score of the K-th document in the top-K, considering the term as a single term query.
- **Number of promotions into top-K:** The number of times some document would make it into the top-K, considering the term as single term query.
- **IDF:** The inverse document frequency of the term.

These term-based features are then aggregated to generate query-based features using three functions: maximum, variance and sum. This process generates a feature set containing 39 query-based aggregated features per query.

We then process the original queries in the query set to record the number of scored postings. This value is independent by the CPU frequency and we can use any $f \in F$. From the execution of the query set, we collect a processing log which contains the number of scored posting for each query in the query set. We use this processing log in the training and validation phase of the predictors.

To train our predictors, we split the feature set and the processing log: 50% of the queries for training and 50% for validation. We use the training set to learn the set of linear regressors π_x, one for each query class. Each regressors take in input the 39 query-based aggregated features from the feature set, and estimates the number of postings scored in the processing log[3]. Note that linear regressors can return negative values for a set of input features. However, the number of scored postings is always a positive quantity. If a regressor returns a negative value, we set its prediction to the minimum between the shortest posting list length for the query terms and 1,000 (the number of retrieved document). Similarly, a linear regressor may return a value that exceeds the sum of the posting lists lengths for a query. Since this is not possible in practice, in such cases we set the prediction to the sum of the posting lists lengths.

Once we have trained the regressors on the training set, we use the validation set to see how predictors perform. Results are reported in Table 5.1. Note that single term queries correspond to a smaller number of postings than queries with more terms. This is expected, as most of the single term queries are associated to a low number of documents. For instance, 2,276 out of 5,000 queries in the validation set are associated to less than 1,000 postings. Also, we notice how we can perfectly predict the number of scored postings for the single term queries. In this case, in fact, MaxScore and WAND cannot prune the posting list and the number of scored postings coincides with the posting list length of the query term. For longer queries, instead, we notice a lower coefficients of determination R^2 which indicate that the predictors are less able to estimate the number of scored postings. For this reason, we use the RMSE ρ_x computed in the validation phase to correct the value of the predictors (as explained in Section 5.3.2).

As explained in Section 5.3.2, we use the RMSE ρ_x computed in the validation phase to correct the value of the predictors. This will provide more conservative predictions to use into OYDS. The result of the training and validation phases is a set of predictors $\Pi = \{\tilde{\pi}_1, \tilde{\pi}_2, \ldots\ldots, \tilde{\pi}_{6+}\}$.

5.4.2 Training processing time predictors

OYDS produces processing speeds that need to be mapped into CPU core frequencies. For this purpose, we process the 60,000 queries set described in Section 5.4.1 to collect

[3]Predictions take approximately less than 0.2 ms on average. This includes the time for computing query features, while term features are computed offline and stored in main memory.

Table 5.1: Mean total number of postings (MTP) and, for both MaxScore and WAND, mean number of scored postings (MSP), root mean squared error (ρ_x) produced by the modified QEPs and coefficient of determination R^2, for each query class (QC). Posting numbers are rounded to thousands.

QC	MTP	MaxScore			WAND		
		MSP	ρ_x	R^2	MSP	ρ_x	R^2
1	31	31	0	1.00	31	0	1.00
2	3,220	1,119	720	0.88	960	746	0.86
3	7,271	1,087	708	0.70	563	569	0.61
4	11,012	1,487	867	0.59	578	397	0.52
5	14,744	1,952	974	0.60	684	307	0.68
6+	21,012	2,769	1,267	0.70	877	464	0.75

the number of scored postings and the processing times of each query. From these data, we learn a set of single-variable linear regressors σ_x^f that estimate the processing time of a query given the number of its scored postings.

The processing time of a query is influenced by the CPU core frequency but also by the workload faced by the query processing node. In fact, high workloads increase the contention among the query servers (i.e., processing threads) for the main memory and the processor caches. This contention increases the time required to process a query. We want our regressors to predict processing times that match high workload conditions. This is a worst-case choice that will lead to higher energy consumption when the query processing node deals with low workloads. However, we expect to miss less query deadlines when the query processing node faces high query volumes. We process the 60,000 query set sending it to the processing node at the rate of 100 queries per second since this rate ensure than our node is constantly busy processing queries, simulating a high query workload. We process the query set 15 times, one for each frequency $f \in F$. We hence obtain 15 different processing logs reporting the number of scored postings and the processing time for each query in the query set.

Again, we divide the queries into six classes (see Sec. 5.4.1). For each query class and each frequency f, we learn a single-variable linear regressor σ_x^f. To learn these regressors, we split each processing log for training and validation: 50% of the logs are used for training the regressors, the remaining 50% is used to validate them. We use the validation set to check how well the predictors perform after the training phase, measuring their RMSE ρ_x^f and the coefficient of determination R^2.

Results are reported in Table 5.2. As expected, the mean processing times decrease by increasing the CPU frequency. However, the mean processing times for the first query class are sensibly lower than for the other classes. This is due to the many single term queries associated to short posting lists, as illustrated in Section 5.4.1. We notice that the processing times are lower when using MaxScore rather than WAND. This confirms the findings in [34, 41, 87], where MaxScore outperforms WAND for memory-resident indexes. Moreover, for the MaxScore strategy we obtain lower RMSE and higher R^2 value than for WAND, meaning that the predictors for MaxScore produce better estimates of the processing time of a query. Presumably, this is due to the fact that the WAND strategy prune posting lists more aggressively than MaxScore (see Table 5.1) and the linear regressors are not able to capture this behavior. For the sake of brevity, Table 5.2 shows the results for only 6 core frequencies. Similar observations can be made for the remaining ones.

TABLE 5.2: Mean processing time (M, in ms), root mean squared error (ρ_x^f) produced by the σ_x^f regressors and coefficient of determination R^2 for each query class (QC) for the MaxScore and WAND retrieval strategies.

QC	0.8 GHz			1.4 GHz			2.0 GHz			2.5 GHz			2.9 GHz			3.5 GHz		
	M	ρ_x^f	R^2	M	ρ_x^f	R^2	M	ρ_x^f	R^2	M	ρ_x^f	R^2	M	ρ_x^f	R^2	M	ρ_x^f	R^2
MaxScore																		
1	9	7	0.99	5	3	1.00	4	2	1.00	3	2	0.99	3	2	0.99	2	2	0.99
2	344	48	0.99	197	21	1.00	141	17	0.99	115	14	0.99	100	13	0.99	83	16	0.99
3	386	52	0.98	224	27	0.99	161	20	0.99	131	16	0.98	114	15	0.98	96	15	0.97
4	549	60	0.98	319	25	0.99	229	20	0.99	186	16	0.99	162	15	0.99	136	16	0.98
5	737	67	0.98	428	29	0.99	307	22	0.99	248	18	0.99	216	16	0.99	181	18	0.98
6+	1,088	150	0.97	633	95	0.97	451	66	0.97	363	52	0.97	316	46	0.97	262	37	0.97
WAND																		
1	14	18	0.98	8	17	0.96	6	10	0.96	5	6	0.98	4	5	0.98	4	10	0.91
2	560	249	0.95	314	145	0.94	225	103	0.94	181	80	0.94	159	72	0.94	132	63	0.93
3	499	235	0.84	281	134	0.83	205	99	0.83	164	78	0.83	145	69	0.83	119	57	0.83
4	675	283	0.77	383	157	0.77	276	114	0.76	223	91	0.77	196	81	0.77	162	67	0.77
5	915	294	0.82	522	175	0.81	381	128	0.82	306	107	0.79	266	89	0.81	216	74	0.80
6+	1,388	620	0.86	780	310	0.88	562	204	0.90	453	179	0.88	393	155	0.88	326	125	0.88

As explained in Section 5.3.3, we use the RMSE R_x^f computed in the validation phase to compensate the predictors' estimates. The result of the training and validation phases is a set of predictors $\Sigma = \{\tilde{\sigma}_1^f, \tilde{\sigma}_2^f, \ldots\ldots, \tilde{\sigma}_{6+}^f\}$.

5.4.3 Measuring energy consumption and latency

We now describe the experimental setup for measuring the CPU energy consumption and the latency for processing a stream of queries on a query processing node. We here focus on the tail latency since it is assumed to be a better performance indicator than the mean/median latency for web search engines [33]. In fact, measuring the tail latency, we can affirm that most of the requests are served within the measured time interval. We require that queries are processed with a certain tail latency. We experiment with a required tail latency of 500 ms and 1,000 ms. The first value represents a scenario where we want to promptly answer the queries, while the second represents the case where we are willing to wait more time to obtain query results. In fact, search engine users are likely to not notice response delays up to 500 ms, while they are very likely to perceive delays higher than 1,000 ms [5]. In PESOS we can impose the tail latency constrain setting $\tau = \{500, 1,000\}$ ms, i.e., requiring that queries are processed within τ ms since their arrival. We test different latency requirements to observe if PESOS can produce energy savings while meeting the required tail latency. The query processing is performed using the MaxScore and the WAND retrieval strategies, to understand how PESOS behaves when different retrieval strategies are deployed. Also, we test PESOS with predictors corrected using their RMSE (as discussed in Sec. 5.3.2 and 5.3.3), and without any correction. We will refer to the first configuration as *time conservative* (TC) and to the second as *energy conservative* (EC). In the TC configuration, we are likely to over-estimate the processing volume and time of some queries, requiring higher core frequencies. However, we also expect to miss less query deadlines hence producing lower tail latencies. In the EC configuration, instead, we use predictors without any correction which should lead to lower core frequencies and produce higher energy savings. Comparing the two

TABLE 5.3: Distribution of queries across the various query classes for the synthetic and the realistic query sets.

	1	2	3	4	5	6+
Synthetic	5,644	17,871	18,913	10,828	4,331	2,413
Realistic	51,553	161,973	171,016	98,001	39,998	22,177

configurations, we want to understand if acceptable tail latencies are achievable even without predictors correction.

To perform our measurements, we carry out two different kinds of experiment. Firstly, we observe the behavior of PESOS under a synthetic query workload. For this purpose, we send a stream of 60,000 unique queries from the MSN2006 log to the processing node. Table 5.3 shows the number of queries for each query class, with an average of ∼3 terms per query. This value reflects the average query length observable on the original MSN2006 log. To test the robustness of PESOS, we experiment with different query arrival rates, i.e., {5, 10, 15, 20, 25, 30, 35} query per second (QPS) sent to the processing node[4]. The second kind of experiment aims to observe the behavior of PESOS under a realistic query workload. For this, we process 544,718 unique queries from the MSN2006 query log following the actual query arrivals of the second day of the query log. Table 5.3 reports the number of queries for each query class, while Figure 5.3 show the number of query arrivals during the day. For both query workloads, we process unique queries to avoid caching mechanism that could compromise the evaluation of the experiment results. Nevertheless, for the realistic query workload we are still processing the same number of queries reported in the second day of the MSN2006 query log to reflect a realistic query traffic.

Finally, we compare the energy consumption and the tail latency of PESOS against three baselines, namely `perf`, `power`, and `se-cons`. `perf` and `power` are provided by

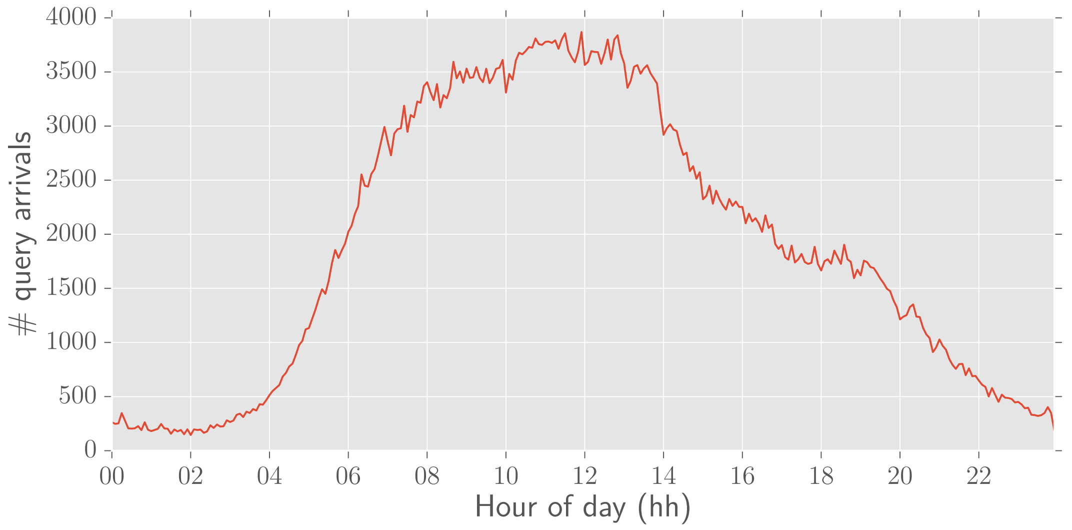


FIGURE 5.3: Query arrivals for the second day of the MSN2006 query log, aggregated every 5 minutes.

[4]Note that the τ and QPS values can be rescaled by considering smaller inverted indexes, for instance when the index is partitioned across multiple query processing nodes.

the `intel_pstate` driver [103]. The `perf` policy simply uses the highest core frequency to process queries and then race to an idle state. The `power` policy, instead, selects the frequency for a core according to its utilization. High frequencies are selected when a core is highly utilized. Conversely, lower frequencies are selected when a core is lowly utilized. Differently, the `se-cons` policy bases its decisions upon the utilization of a query server rather than on the utilization of a CPU core. The utilization of a query server is computed as the ratio between the query arrival rate and service rate. The frequency of a core is then throttled if the server utilization is above 80% or below 20%, to produce a utilization of 70%. The `se-cons` policy executes every 2 seconds. We select these parameter settings to achieve the best energy savings while maintaining acceptable latencies, reflecting those used in Chapter 4.

With these experiments we want to address the following research questions:

- RQ1: Does PESOS meet the required tail latencies?
- RQ2: Does PESOS help reduce the CPU energy consumption of a query processing node?
- RQ3: Is prediction correction necessary to achieve acceptable tail latencies?
- RQ4: How does PESOS behave using different retrieval strategies, with different prediction accuracies?

We measure the 95-th percentile tail latency of the processing node to answer our first research question. The 95-th percentile tail latency is used to measure the effects of power management mechanism on the responsiveness of search systems in [71, 80]. To answer the second research question we measure the energy consumption of the CPU using the Mammut library which relies on the Intel Running Average Power Limit (RAPL) interface [30]. The RAPL component performs actual measurements of the energy consumption in Haswell processors. Hackenberg et al. show the reliability of such measurements [52], and the RAPL interface is used in other works to measure the energy consumption of CPUs [29, 31]. We experimentally confirm the reliability of RAPL in Appendix C. Finally, to address the third research question we compare the performance of our approach with and without prediction corrections. We compare the performance of PESOS with MaxScore and WAND to answer the last research question. All experiments are conducted using the query processing node described at the beginning of this section.

5.4.4 Other experimental setup details

All the experiments introduced in this section are conducted using the Terrier IR platform [76], a Java search engine. The platform is hosted on a dedicated server equipped with an Intel i7-4770K CPU, a member of the Haswell product family. This CPU has eight logical cores that are mapped into four physical cores, thanks to hyperthreading. Recall that PESOS selects the most appropriate frequency to process a query on a per-core basis. Therefore, logical cores can be instructed to operate at different logical frequencies. This can result in conflicts when mapping multiple logical frequencies into a single physical operational frequency. To avoid such conflicts, we decide to not exploit hyperthreading in our experiments. Anyway, PESOS can be easily adapted to exploit hyperthreading as follows: each logical core selects a frequency according to PESOS; the physical core frequency is then set to the maximum one among those required by its logical cores.

Since PESOS selects which frequency to use to process a certain query on a core, it is also necessary that the corresponding processing threads do not migrate from that core to another. For this reason, query processing threads are pinned to different CPU physical cores. This is achievable in Java by using the Thread Affinity library [28]. Thread Affinity is a Java library which permits binding threads to core from the Java code. It relies on Java Native Access (JNA), a library which provides Java programs easy access to native shared libraries without using the Java Native Interface (JNI).

Finally, since Terrier is a Java-based platform, we developed a library[5] which permits to select which frequency to use on a CPU core from the Java code. Our library is build on top of the `acpi_cpufreq` Linux driver [17]. The driver exposes a set of files in the sysfs which can be used to control the P-state of each core by simply reading/writing them, if permission are granted. On average, our library takes less than 10 microseconds to modify the operating frequency of a core. This value is below the acceptable query processing time.

5.5 Results

In this section we discuss the results of our experiments. We firstly describe the results relatively to the experiments conducted with synthetic query workloads. Then, we illustrate the results obtained using the realistic query workload.

5.5.1 Synthetic query workload results

We begin by analyzing the behavior of `perf` and `power`. We recall that `perf` always uses the maximum available CPU core frequency, while `power` is a utilization-based policy which throttles a CPU core frequency accordingly to its utilization. Both `perf` and `power`, however, do not permit to impose the required tail latency of a query processing node. From Table 5.4 we can observe that, when MaxScore is deployed, `perf` meets the 500 ms tail latency requirement up to 30 QPS, while the 1,000 ms tail latency requirement is always satisfied. When WAND is used, instead, `perf` satisfies the 500 ms tail latency up to 20 QPS, and the 1,000 ms tail latency up to 30 QPS. We explain this difference by recalling that WAND provides longer response times than MaxScore (see Table 5.2). With respect to tail latencies, we observe a similar behavior between `perf` and `power`. This is expected since, as the query arrival rate increases, the CPU cores utilization increases as well, leading `power` to select high core frequencies and hence behaving like `perf`. In terms of energy savings[6], Table 5.5 shows little differences between the two baselines. Some energy savings are provided by `power` at low QPS, from ∼2% in the case of WAND up to ∼5% for MaxScore, at the cost of higher tail latency. For high query arrival rates, `power` can be even detrimental, increasing the energy consumption of the system. We explain this behavior with the longer query processing times and the overhead introduced by the policy, i.e., the CPU cores spend more time busy doing computations, hence consuming more energy.

Regarding the other baseline, we observe in Table 5.4 that `se-cons` satisfies the 500 ms tail latency only for moderate QPS (from 15 to 25) when MaxScore is deployed, and only for 20-25 QPS with WAND. Again, this is due to the better performance of MaxScore over WAND. When considering a tail latency of 1000 ms, we observe that `se-cons` meets the latency requirement from 10 to 35 QPS with MaxScore and from

[5]`https://github.com/catenamatteo/dvfs4j`.

[6]Energy consumption decreases as the query arrival rate increases, since experiments take less time to complete.

TABLE 5.4: MaxScore (top) and WAND (bottom) tail latencies (95th-tile, in ms) of baselines, time conservative (TC), and energy conservative (EC) PESOS for different synthetic query workload (QPS).

QPS	Baselines			PESOS			
				$\tau = 500$ ms		$\tau = 1,000$ ms	
	`perf`	`power`	`se-cons`	TC	EC	TC	EC
MaxScore							
5	342	360	1,019	446	573	809	980
10	344	344	667	431	536	759	894
15	341	346	442	428	509	703	833
20	362	364	393	415	489	685	832
25	402	400	411	446	500	701	842
30	479	498	515	522	563	835	948
35	657	715	687	725	731	1,174	1,287
WAND							
5	378	399	1,060	399	538	649	896
10	380	382	714	389	510	615	813
15	391	396	519	401	490	586	757
20	437	436	457	439	502	585	765
25	527	537	546	534	569	627	793
30	821	835	787	821	867	884	1,035
35	2,696	3,091	2,831	3,211	3,585	2,667	3,318

10 to 30 QPS with WAND. In general, we can conclude that `se-cons` produces latency violations when the query arrival rate is particularly low or high. We explain this behavior by recalling that `se-cons` requires to tune several parameters which drive its decisions about frequency scaling. In our experiments we use a setting aimed to produce the best energy savings and acceptable latencies. However, our results suggests that a single parameter setting is not sufficient for `se-cons` to perform well under a wide range of query arrival rates. With respect to energy consumption, Table 5.5 shows that `se-cons` provides substantial energy savings with respect to `perf` at low QPS (up $\sim$ 45% with Maxscore and $\sim$ 40% with WAND). However, when the query arrival rate increases, `se-cons` can consume more energy. Again, we explain this behavior with the longer query processing times and the overhead introduced by the policy.

We now discuss the results for PESOS when using $\tau = 500$ ms and $\tau = 1,000$ ms. For the time conservative configuration, Table 5.4 shows that PESOS satisfies the 500 ms tail latency requirement from 5 to 20 QPS when using WAND and up to 25 QPS when using MaxScore. For the 1,000 ms tail latency requirement, in the time conservative configuration PESOS meets the required latency up to 30 QPS for both retrieval strategies. These results are similar to what reported for the `perf` policy. Relatively to our first research question (RQ1), we can state that PESOS is able to

TABLE 5.5: Energy consumption (KJ) of baselines, time conservative (TC), and energy conservative (EC) PESOS, with energy savings w.r.t. `perf` for different synthetic query workload (QPS).

QPS	Baselines			PESOS			
				$\tau = 500$ ms		$\tau = 1,000$ ms	
	`perf`	`power`	`se-cons`	TC	EC	TC	EC
				MaxScore			
5	92.79	87.78 (-5.40%)	51.06 (-44.97%)	69.95 (-24.62%)	61.34 (-33.89%)	47.43 (-48.89%)	42.56 (-54.13%)
10	83.51	81.35 (-2.58%)	58.32 (-30.16%)	65.38 (-21.71%)	57.30 (-31.39%)	47.36 (-43.29%)	44.81 (-46.34%)
15	77.78	77.54 (-0.31%)	74.33 (-4.44%)	64.35 (-17.26%)	57.26 (-26.37%)	50.22 (-35.43%)	48.74 (-37.34%)
20	75.37	75.34 (-0.05%)	75.01 (-0.48%)	62.21 (-17.46%)	59.42 (-21.17%)	52.35 (-30.55%)	52.65 (-30.15%)
25	72.75	73.09 (0.47%)	74.23 (2.03%)	65.77 (-9.59%)	62.57 (-13.99%)	56.46 (-22.39%)	56.74 (-22.01%)
30	70.74	71.43 (0.98%)	72.61 (2.64%)	66.50 (-6.00%)	65.06 (-8.04%)	62.42 (-11.76%)	65.01 (-8.10%)
35	69.78	71.51 (2.48%)	71.53 (2.51%)	68.46 (-1.89%)	66.89 (-4.14%)	70.02 (0.35%)	68.70 (-1.55%)
				WAND			
5	106.49	104.28 (-2.07%)	64.11 (-39.80%)	93.83 (-11.89%)	76.03 (-28.60%)	67.38 (-36.72%)	56.48 (-46.96%)
10	96.62	95.25 (-1.42%)	74.13 (-23.28%)	88.01 (-8.91%)	74.66 (-22.73%)	67.11 (-30.54%)	60.19 (-37.70%)
15	91.55	91.98 (0.46%)	87.27 (-4.66%)	84.56 (-7.64%)	75.80 (-17.21%)	68.39 (-25.30%)	64.27 (-29.81%)
20	89.34	89.31 (-0.04%)	89.27 (-0.08%)	83.49 (-6.55%)	78.44 (-12.20%)	72.11 (-19.29%)	70.72 (-20.84%)
25	85.81	86.59 (0.91%)	87.17 (1.58%)	83.69 (-2.47%)	79.32 (-7.56%)	75.96 (-11.48%)	73.92 (-13.86%)
30	85.27	86.38 (1.31%)	85.85 (0.68%)	84.37 (-1.05%)	82.03 (-3.80%)	80.82 (-5.22%)	80.03 (-6.14%)
35	84.58	84.86 (0.34%)	85.72 (1.35%)	84.70 (0.15%)	84.88 (0.36%)	82.72 (-2.20%)	84.15 (-0.50%)

meet the required tail latencies for the same query workloads sustainable by a system which operates at maximum CPU core frequency.

In terms of energy savings, Table 5.5 shows that PESOS markedly reduce the energy consumption of the query processing node's CPUs. In the time conservative configuration, PESOS can reduce the energy consumption up to ∼25% when using MaxScore and up to ∼12% when using WAND. We explain the better results achieved with MaxScore with the higher accuracy of its processing time predictors compared to the ones for WAND (see Table 5.2). We also notice that energy savings diminish as the query arrival rate increases, as there are fewer opportunities for PESOS to use low core frequencies without violating query deadlines. Relatively to our second research question (RQ2), the results in Table 5.5 show that PESOS actually permits to reduce the CPU energy consumption of a query processing node. In most cases, these energy savings are higher than those provided by the state-of-the-art `power` and `se-cons` policies. This indicates that application-dependent information leveraged by PESOS, such as the state of the query queues and the query efficiency predictors, are a better input for managing the CPU cores frequencies than the cores or query servers utilizations. Also, an important role is played by the τ parameter, which permits to set the required tail latencies rather than processing the queries at maximum speed as in `perf`, which does not take into account latency requirements.

We now analyze the performance of PESOS in the energy conservative configuration, i.e., when we do not correct the query efficiency predictors using their RMSE. Table 5.4 shows that, for both retrieval strategies, PESOS misses the 500 ms tail latency requirement. This answer our third research question (RQ3): predictors correction is necessary to meet the latency requirements. However, we highlight that the reported latency violations are limited: for the same QPS values for which the time conservative configuration meets the 500 ms tail latency requirement, the energy conservative configurations violates the requirement by up to ∼8% with WAND and up to ∼15% with MaxScore. Additionally, we notice higher energy savings compared to the time conservative configuration (see Table 5.5). When $\tau = 500$ ms, the energy conservative configuration reduces the energy consumption of the CPU node by ∼29% in the case of WAND and by ∼34% in the case of MaxScore for low QPS. In Table 5.4 we can observe that the 1,000 ms tail latency requirement is met up to 30 QPS when MaxScore is applied, and up to 25 QPS when WAND is used. This suggests that predictors' correction becomes less relevant as the latency requirement increases. Remarkably, the energy conservative configuration basically halves the energy consumption of the CPU node for 5 QPS when $\tau = 1,000$ ms (see Table 5.5).

Finally, to answer our last research question (RQ4), we compare the behavior of PESOS while deploying MaxScore and WAND. In general, PESOS shows better results with MaxScore. In fact, the tail latency requirements are met for slightly higher QPS values compared to WAND. Also, PESOS shows higher energy savings when the MaxScore retrieval strategy is applied. We explain this behavior with the faster response time provided by MaxScore and by the higher precision of its processing time predictors.

5.5.2 Realistic query workload results

Now we describe the results of the experiments conducted processing the realistic query workload. In this subsection we will not investigate research question RQ4 as for these experiments we use only the MaxScore retrieval strategy, which provided the best results in Section 5.5.1. Firstly, we will analyze the performance of the three baselines. Then, we will discuss the results obtained by PESOS in the time

Table 5.6: CPU energy consumption (KJ) of the power management approaches for processing a day of query log, and the gain w.r.t. `perf`.

	Energy (KJ)	Gain (%)
`perf`	790.40	–
`power`	759.42	-3.92%
`se-cons`	575.49	-27.19%
PESOS (TC, $\tau = 500$ ms)	601.67	-23.88%
PESOS (EC, $\tau = 500$ ms)	531.10	-32.81%
PESOS (TC, $\tau = 1,000$ ms)	443.73	-43.86%
PESOS (EC, $\tau = 1,000$ ms)	412.06	-47.87%

conservative configuration. Finally, we will study the performance of PESOS in the energy conservative configuration.

Figure 5.4 reports the tail latencies of the tested approaches during the day. As expected, `perf` provides lower latencies than the other approaches. Unsurprisingly, `perf` exhibits also the higher CPU energy consumption as it always uses the maximum core frequency (see Tab. 5.6). In terms of tail latency, `power` behaves similarly to `perf` during midday but exhibits higher latencies at the beginning and at the end of the day. This behavior is explained in Figure 5.6 (top). During midday, the CPU cores are highly utilized due to the higher number of query arrivals. In response to high core utilization, `power` selects the maximum core frequency as in `perf`. During the rest of the day, instead, the query arrivals decrease and the CPU cores are less utilized. Therefore, `power` selects lower core frequencies which explain longer latencies. For the same reasons, `power` provides limited energy savings compared to `perf`, reducing the CPU energy consumption by less than 4% as reported in Table 5.6. Figure 5.5 illustrate the energy reductions of `power` with respect to `perf` during the day. When `power` is applied, we can observe energy savings only at the beginning and at the end of the day, when `power` selects lower core frequencies as shown in Figure 5.6 (top). In these periods, the CPU consumes ~20% less energy with respect to `perf`. However, during midday `power` does not provide any energy saving. Again, this is due to the high utilizations showed by the CPU cores during midday In this situation, `power` selects the maximum core frequency, behaving like `perf` and consuming the same amount of energy.

Table 5.6 shows that `se-cons` can reduce by ~27% the CPU energy consumption with respect to `perf`. As shown in Figure 5.5, energy consumption can be reduced by ~45% during periods of low query workloads. However, such energy savings come at the price of high latencies (see Fig. 5.4). Indeed, `se-cons` exhibits tail latencies that are above 500 ms during midday, and above 1,000 ms during the rest of the day. In fact, `se-cons` relies on low core frequencies most of the time (see Fig. 5.6 (middle)), hence producing long query response times. It starts selecting higher core frequency only during midday, when the query servers utilizations increases due to the higher query workload. In fact, we can notice in Figure 5.4 how tail latencies reduces during midday. The results reported in this section confirm those presented in Section 5.5.1, where `se-cons` poorly behaves in presence of low query arrival rates. Indeed, `se-cons` requires a careful parameter tuning, and a static parameter setting cannot efficiently cope with query arrivals rate that widely varies throughout the day.

Regarding time conservative PESOS, we can observe in Figure 5.4 that the 500 ms tail latency requirement is successfully met, with very few violations. Similarly,

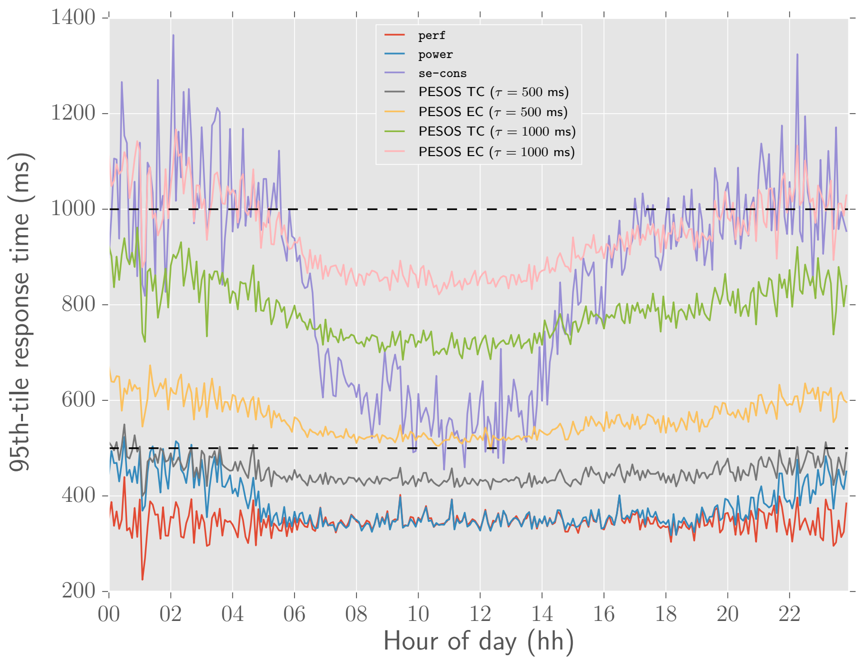


FIGURE 5.4: Tail latencies during a day, aggregated every 5 minutes.

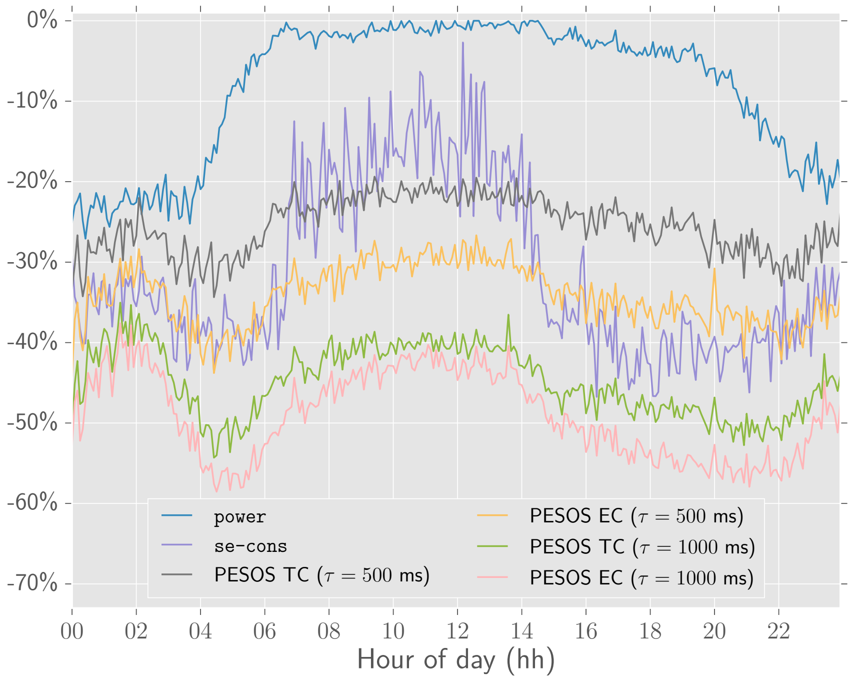


FIGURE 5.5: CPU energy reductions w.r.t `perf`, aggregated every 5 minutes.

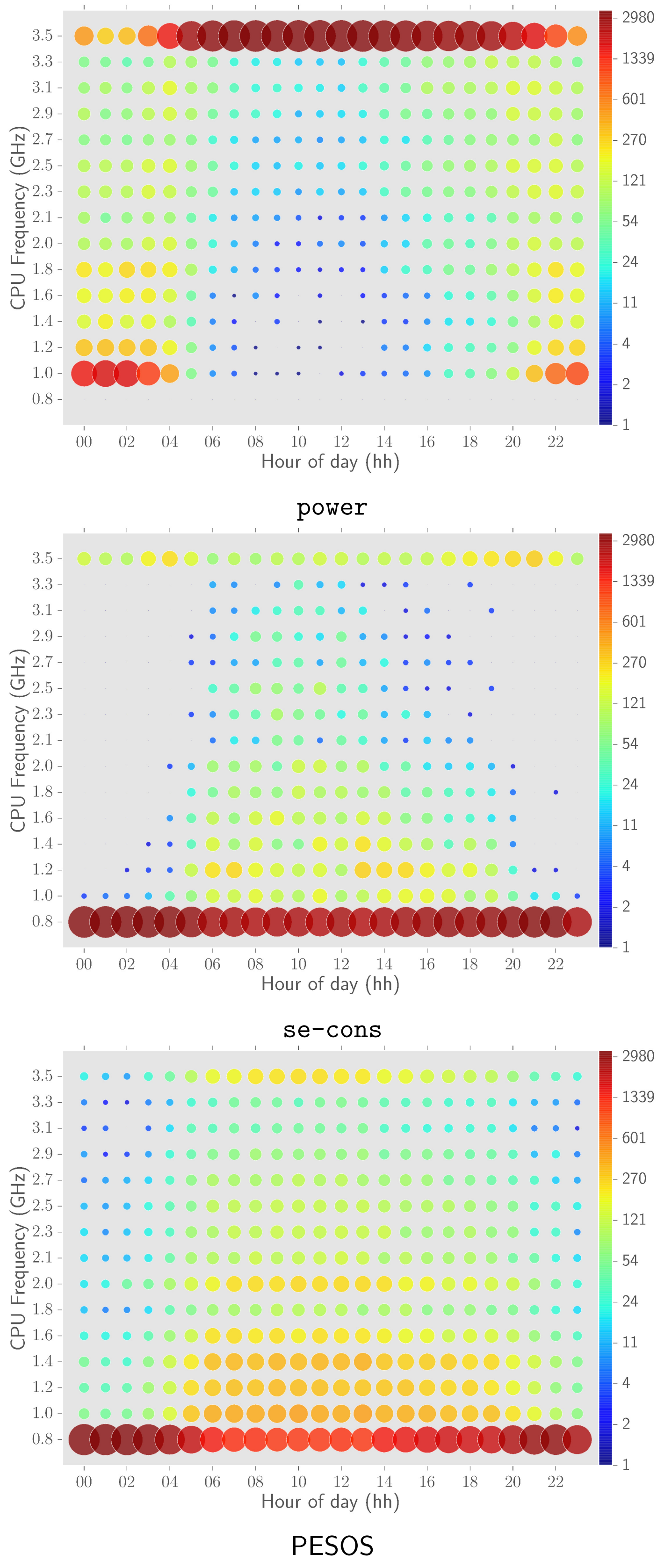


FIGURE 5.6: Number of times `power` (top), `se-cons` (middle) and time-conservative ($\tau = 500$ ms) PESOS (bottom) select frequencies on one of the CPU cores during the day, sampled every second.

the time conservative configuration is able to meet the 1,000 ms tail latency requirement, remaining well below the required threshold. Relatively to our first research question (RQ1), we conclude that PESOS can successfully meet the required tail latency when the time conservative configuration is applied, even for a realistic query workload. At the same time, PESOS shows significant energy savings with respect to `perf`, as reported in Table 5.6. In fact, with a 500 ms tail latency requirement, time conservative PESOS reduces the CPU energy consumption by ∼24%, and by ∼44% when we impose a 1,000 ms tail latency requirement. As shown in Figure 5.5, such energy savings are present during the whole day, up to ∼30% under the 500 ms tail latency requirement, and ∼50% for the 1,000 ms tail latency requirement. These energy savings are possible thanks to the application-level information exploited by the PESOS algorithm, such as the states of the query queues and the query efficiency predictions. Also, an important role is played by the τ parameter, which permits to set the required tail latency. As we can see from Figure 5.6 (bottom), this information permits PESOS to select lower core frequencies more often than `power`, which takes its frequency scaling decision relying only on the CPU cores utilizations. High core frequencies are selected by time conservative PESOS only in limited cases during midday, when the query load is more intense. Relatively to our second research question (RQ2), we can then conclude that PESOS successfully reduces the CPU energy consumption of a query processing nodes, providing much higher energy savings than the `power`. At the same time, PESOS provides energy savings comparable to those produce by `se-cons`, while incurring in many less latency violations.

We now analyze the results for PESOS in its energy conservative configuration. As shown in Figure 5.4, energy conservative PESOS does not satisfy the tail latency requirements. However, we notice that the tail latency of energy conservative PESOS approaches the 500 ms requirement during midday, when the query load is more intense and the query queues are populated with a higher number of queries than in other periods of the day. In the PESOS algorithm, this results in critical intervals of high intensity (see Alg. 4) which lead PESOS to select higher core frequencies, hence reducing the tail latency of the system. We can observe the same effects when we impose a 1,000 ms tail latency requirement. In this case, the energy conservative configuration violates the requirement at the beginning and at the end of the day, when the query workload is not intense. On the contrary, the 1,000 ms tail latency requirement is met during midday in correspondence of a high query arrival rate. While violating the tail latency requirements, the energy conservative configurations provides the highest energy savings as reported in Table 5.6. When we impose a tail latency requirement of 500 ms, energy conservative PESOS reduces the CPU energy consumption by almost 33% compared to `perf`. The energy savings reach ∼48% when the tail latency requirement is set to 1,000 ms. Such savings are present during the whole day, as illustrated in Figure 5.5, up to ∼40% under the 500 ms tail latency requirement, and ∼60% for the 1,000 ms tail latency requirement. Interestingly, we notice a larger energy saving gap between time conservative and energy conservative PESOS when $\tau = 500$ ms than when $\tau = 1,000$ ms. This is particular evident in Figure 5.5, where the curves relative to the two configurations almost coincides. This is surprising, as time conservative PESOS is likely to over estimate the processing volumes and times for some queries, selecting higher core frequencies and consuming more energy. However, this behavior can be due to the longer time budgets available to process queries under the 1,000 ms latency constraint, which still permits time conservative PESOS to select lower core frequencies to process queries. Regarding our third research question (RQ3), we can then conclude that predictors correction is necessary to meet the required tail latencies and overall time conservative PESOS is a better choice when processing a realistic query workload.

5.5.3 Additional results on latency

Tail latency is assumed to be a better performance indicator for web search engines than median and mean latency [33]. Nevertheless, the reader can be interested also in these metrics. Therefore, we report in the following the median and mean response time measured in our experiments.

Mean and median latencies under the synthetic query workload

We here focus on the mean latency obtained processing the queries using the MaxScore matching algorithm. From Table 5.7, we observe that the lowest mean latency is obtained by `perf`, which processes all queries using the maximum core frequency. `power` exhibits mean latencies similar to `perf`. However, for 5 and 35 QPS, we observe that `power` mean latency is slightly higher than `perf`'s. We recall that `power` bases its frequency throttling decisions depending on the cores' utilization. Since low QPS implies low resource utilization, `power` tends to select low core frequency when experiencing scarce workloads. This behavior explains the higher mean latencies w.r.t. `perf` at 5 QPS. Instead, the higher mean latency reported at 35 QPS can be explained by the overhead introduced by the `power` algorithm.

For `se-cons`, we observe that the mean latencies are higher than `perf`, especially for low and high QPS. In fact, `se-cons` is a utilization-based policy, which base its frequency throttling decisions upon the utilization of the query servers. Therefore, `se-cons` incurs in the same shortcomings of `power`. Additionally, `se-cons` requires some parameter tuning, which definitively plays a role here. In our experiments, we used the same parameter settings adopted in Chapter 4 to produce the best energy savings and latencies. However, results suggest that a single, static, parameter setup is insufficient for `se-cons` to cope with a wide range of query arrival rates.

We now discuss the results regarding PESOS with MaxScore. From Table 5.7, we can observe higher mean latencies than `perf` while using PESOS. However, we recall that PESOS aims to reduce the CPU energy consumption while respecting query deadlines. Deadlines are imposed via the τ parameter, i.e., queries have to be processed within τ ms since their arrival to the query processing node. Indeed, we observe that the mean latency for PESOS is always below the τ value. In fact, PESOS does not try to process queries as fast as possible. Instead, it just aims to process queries by their deadlines. This permits our policy to better exploit low core frequency, providing the superior energy savings reported in Section 5.5.1.

The considerations drawn for the mean latencies and MaxScore are valid also for the median latencies. Similar considerations can be derived for the results with WAND, for which we observe higher latencies.

Mean and median latencies under the realistic query workload

We here report the mean latencies measured for the experiments conducted processing the realistic query workload (see Figure 5.7). For these experiments we use just MaxScore, which provided the best results in Section 5.5.3. The considerations drawn for the mean latencies are valid also for the median ones (see Figure 5.8).

Figure 5.7 shows that `perf` provides lower mean latencies than the other approaches. `power` behaves similarly to `perf` during midday but exhibits higher mean latencies at the beginning and at the end of the day. In fact, CPU cores are highly

TABLE 5.7: Mean and median latencies (in ms) of baselines, time conservative (TC), and energy conservative (EC) PESOS for different synthetic query workload (QPS)

QPS	Baselines						PESOS							
							$\tau = 500$ms				$\tau = 1000$ms			
	`perf`		`power`		`se-cons`		TC		EC		TC		EC	
	median	mean	median	mean	median	mean	median	mean	median	mean	median	mean	median	mean
MaxScore														
5	65	104	72	112	150	282	111	156	138	199	169	257	192	303
10	65	104	66	105	102	186	95	142	117	177	154	237	183	281
15	65	103	66	104	74	128	91	139	111	167	156	228	196	279
20	68	109	68	109	70	115	90	135	113	164	166	232	229	301
25	76	121	76	121	77	124	101	147	123	172	196	254	270	330
30	94	146	99	153	102	158	123	175	149	201	256	346	371	422
35	141	209	162	235	153	224	178	247	204	266	430	503	533	595
WAND														
5	66	113	74	122	157	295	90	136	133	191	143	213	185	285
10	66	113	67	114	99	192	80	127	109	168	122	191	164	255
15	67	116	69	117	77	144	77	126	102	159	118	184	163	244
20	72	128	73	127	73	131	81	134	105	164	123	186	183	260
25	85	150	87	152	87	154	96	156	119	183	138	205	207	280
30	148	245	153	253	140	234	155	248	176	270	206	295	310	393
35	744	969	823	1,078	800	1,024	893	1,137	1,032	1,303	730	962	1,040	1,288

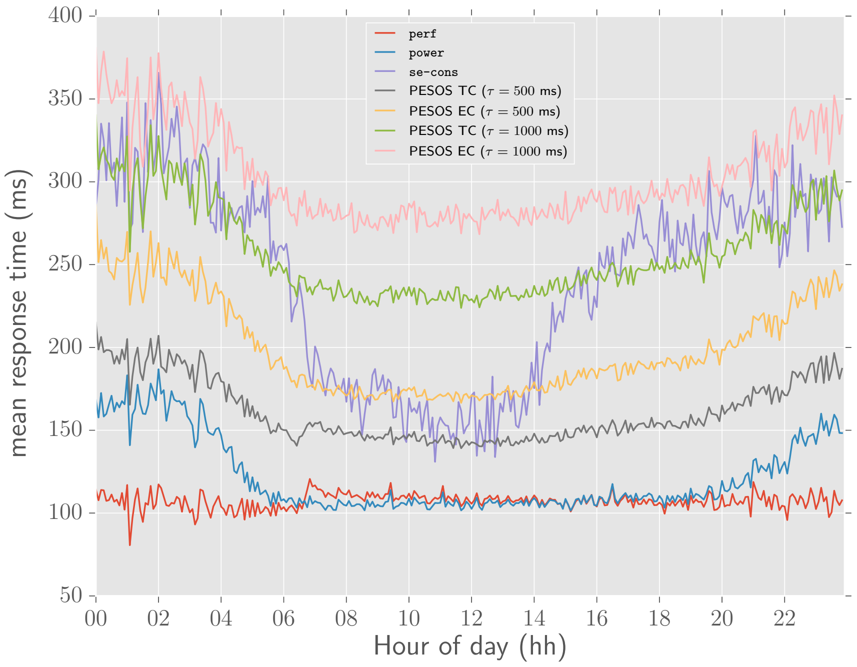


FIGURE 5.7: Mean response times during a day, aggregated every 5 minutes.

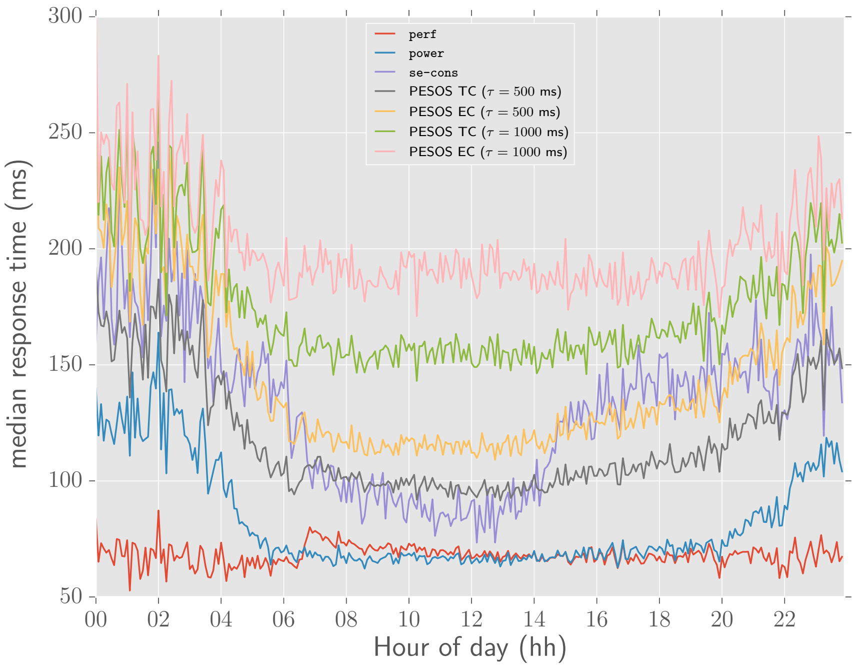


FIGURE 5.8: Median response times during a day, aggregated every 5 minutes.

utilized during midday due to the high number of query arrivals. In response to high core utilization, `power` tends to select high core frequency, producing lower mean latencies. Instead, `power` selects lower core frequency during the rest of the day, since query arrivals are scarcer and the CPU core are less utilized. This explains why `power` mean latencies are higher than `perf` at the beginning and at the end of the day.

From Figure 5.7, we observe that `se-cons` mean response time is higher than `power`'s. However, we notice that the two policies have a similar behavior: they produce higher mean latencies at the beginning and at the end of the day, and lower mean latencies during midday. Indeed, `se-cons` is a utilization-based policy like `power`. Therefore, it tends to use high core frequencies in response to intense query workloads. This explains why `se-cons` mean response time is lower during midday and higher during the rest of the day, when query workload is scarce.

On the other hand, PESOS bases its frequency throttling decision such that each query is served within τ ms since its arrival. Therefore, we observe the lowest mean latencies when we impose $\tau = 500$ ms. In particular, the time conservative configuration produces mean latencies that are generally lower than those produced by `se-cons`. On the other hand, we observe that PESOS shows higher mean latencies than `se-cons` when we impose $\tau = 1000$ ms and we use the energy conservative configuration. However, it is worth to note that, under this setting, the query processing node's CPU consumes just 412.06 KJ, as opposed to the 575.49 KJ consumed by `se-cons`, the 759.42 KJ consumed by `power`, and the 790.40 KJ consumed by `perf`.

5.6 Discussion

In this chapter we proposed the Predictive Energy Saving Online Scheduling (PESOS) algorithm. In the context of web search engines, PESOS aims to reduce the CPU energy consumption of a search server while imposing a required latency on the query response times. PESOS can also be deployed at the inter-data center level (i.e., on a distributed search architecture) since it is completely decentralized. We plan to study its efficacy in such context in a future work.

For each query, PESOS selects the lowest possible CPU core frequency such that the energy consumption is reduced and the deadlines are respected. PESOS selects the right CPU core frequency exploiting two different kinds of query efficiency predictors (QEPs). The first QEP estimates the processing volume of queries. The second QEP estimates their processing times under different core frequencies, given the number of postings to score. Since QEPs can be inaccurate, during their training we recorded the root mean square error (RMSE) of the predictions. In this chapter, we proposed to sum the RMSE to the actual predictions to compensate prediction errors. We then defined two possible configurations for PESOS: *time conservative*, where prediction correction is enforced, and *energy conservative*, where QEPs are left unmodified.

We experimentally evaluated the performance of PESOS using the ClueWeb09B corpus and processing queries from the MSN2006 log applying two different dynamic pruning retrieval strategies: MaxScore and WAND. We compared the performance of PESOS with those of three baselines: `perf`, which always uses the maximum CPU core frequency, `power`, which throttles frequencies according to the core utilizations, and `se-cons`, which throttles frequencies according to the utilization of the search servers. We found that time conservative PESOS was able to meet a required tail latency of 500 and 1,000 ms for the same workload sustainable by `perf`. At the same time, time conservative PESOS was able to reduce the CPU energy consumption of the CPU by ∼12% with WAND up to ∼25% with MaxScore, for which we could train more accurate query efficiency predictors than for WAND. Greater energy savings were observable with energy conservative PESOS, but at the cost of higher latencies. Predictors correction is hence necessary to obtain the required tail latency, still providing significant energy savings. Moreover, we processed a realistic query workload which reflects the query arrivals of one day of the MSN2006 log. We found that time conservative PESOS was able to meet a 500 ms (with very few violations) and a 1,000 ms tail latency requirements, while reducing the CPU energy consumption, respectively, by ∼24% and by ∼44% when compared to `perf`. From the same set of experiments, we reported that `power` can reduce the CPU energy consumption by just ∼4% with respect to `perf`. On the other hand, `se-cons` was able to reduce the CPU energy consumption by ∼27% but incurring in considerable latency violations. We justified the superior `perf` provided by PESOS thanks to the application-level information exploited by our algorithm, such as the knowledge about the state of the query queues and the query efficiency predictions.

Chapter 6

Exploiting Green Energy to Reduce the Operational Costs of Web Search Engines

6.1 Introduction

Commercial web search engines distribute their infrastructures and operations across several, geographically distant, data centers [8, 21, 22, 60]. Each site maintains an index replica of the most recent crawling of the Web [14] and user queries are subsequently processed in the closest available data center to minimize network latencies.

Energy prices play a role in deciding where to build a data center, favoring countries where energy is cheaper. In fact, energy shows both spatial and temporal price variations, with its cost changing during the day due to supply/demand factors [91]. *Query forwarding* has been proposed as a factor to reduce the operational costs of search engines [61, 101]. The main idea is to dispatch queries from the data center that firstly received the request to a different one. In order to reduce the energy expenditure, query forwarding aims to shift the query load towards those data centers which incur in the lowest energy price at every time step.

Moreover, web companies have adopted eco-friendly policies to reduce data centers energy consumption and the relative carbon footprint. These range from designing more energy efficient data centers to increasing the usage of *green energy* – auto-produced or bought from local providers [38, 50, 81].

Green energy comes from resources which are renewable and do not emit carbon dioxide, such as sunlight and wind. On the contrary, *brown energy* is produced using polluting resources like carbon or oil. Since green energy sources are freely available, and thanks to governments incentives, green energy is often cheaper than brown alternatives [47, 100]. However, green energy availability can fluctuate over the day: for example, solar and wind energy production is susceptible to weather conditions. Given that data center energy consumption depends on its usage [10], additional – possibly brown – energy needs to be bought from the energy market, when there is not enough available green energy to sustain the data center workload.

In this chapter we target the energy-related operational costs of large-scale, multi-center web search engines with replicated indexes. Our research is mainly motivated by the following observations:

- green energy is available at different sites in limited quantities;
- green energy usage is assumed to be more convenient than brown energy;
- query workloads of search data centers show spatial-temporal variations, i.e., the workload of a data center varies during the day and some data centers may be under high traffic while others are mostly idle [98].

We propose a new query forwarding algorithm that aims to reduce the overall energy operational costs of a multi-center web search engine. Our approach exploits both the green energy sources available at different sites and the differences in market energy prices. The problem of exploiting green/brown energy to reduce costs when forwarding queries is modeled as a Minimum Cost Flow Problem, taking into account the different and limited processing capacities of data centers, query response time constrains and communication latencies among sites. We evaluate the proposed algorithms using workloads obtained from the Yahoo search engine together with realistic electric price data, and we compare it with a state-of-the-art baseline for query forwarding in distributed search engines. Our results show that, in general, query forwarding plays only a little role in reducing the carbon footprint of current data centers. In other words, we provide an additional evidence of the need for more energy-proportional hardware, i.e., for hardware which consume little or no energy when not being in use [10, 12]. More importantly, we show that our solution obtains energy expenditure reductions that range from $\sim$15% to $\sim$25% with respect to standard multi-center web search engines, outperforming the state-of-the-art.

The rest of the chapter is structured as follows: Section 6.2 proposes a model for a distributed search engine, its energy consumption and its operational costs, while Section 6.3 exploits such models to design a query forwarding algorithm leveraging the green energy available to the search engine. Section 6.4 illustrate our experimental setup and Section 6.5 reports on the results of our comprehensive evaluation. Finally, the chapter concludes in Section 6.6.

6.2 Problem statement

We model the infrastructure of a search engine as a geographically distributed system. We assume that the underlying systems are able to communicate and exchange workload using *query forwarding*, which we aim to leverage in conjunction with different pricing of electricity and availability of green energy around the world. The main goal is to use optimally the amount of green energy available at different sites to minimize the operational costs and, as a side effect, the carbon footprint of the whole infrastructure. The model analytically captures the global state of a search engine infrastructure through a set of state variables. We will implicitly assume that our model is statically valid during a fixed time length or *time slot* Δt.

Search engine model We model a geographically distributed search engine as a set of data centers $\mathcal{D} = \{D_1, \dots, D_N\}$. These data centers are placed in different and distant locations across the planet. Each data center D_i is a pair (F_i, B_i), composed by a frontend F_i and a backend B_i (Figure 6.1).[1] A frontend acts as a requests router: it receives user queries and decides to whether forward them to its corresponding local backend or to some remote backend. In practice, this implies that every frontend is connected through the network to every backend. A backend is composed by several server clusters, which perform the computation required to process an incoming query. Once the results are computed, these are sent back to the frontend that received the original request, which will deliver the results back to the user who issued the query.

Each frontend F_i collects queries from users geographically close to data center D_i. These users act like a query source S_i sending to F_i a certain number of search request a_i over the time slot Δt. The number of queries submitted varies throughout

[1]Free indexes are assumed to run from 1 to N if not specified otherwise.

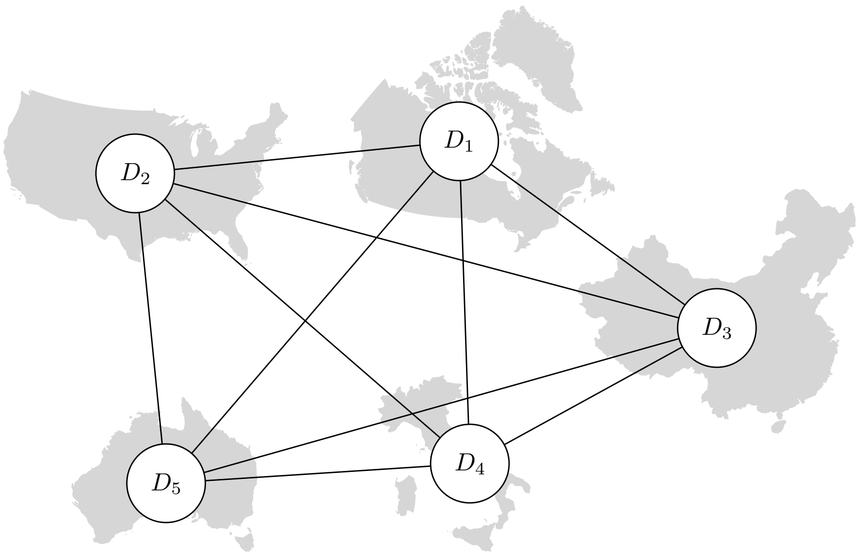


FIGURE 6.1: Example of a web search engine infrastructure model.

the day, being higher at daytime than at nighttime [98]. Due to timezone differences, in the same instant, data centers will experience different query workloads.

After receiving a query, F_i may decide to forward this request to any other backend B_j or to process it locally. Our goal is to determine x_{ij}, i.e., how many search requests from frontend F_i are routed to backend B_j.

In this formulation, we enforce that the following three balance constraints hold.

1. Forwarded search requests can not be negative nor fractional, i.e., for all i and j we must have $x_{ij} \in \mathbb{N}$.
2. The sum of search requests dispatched by frontend F_i must correspond to the search requests received from the source S_i, i.e., for all i we must have:
$$\sum_{j=1}^{N} x_{ij} = a_i \ . \tag{6.1}$$
3. At each time slot, the backend B_j processes y_j queries, which come from the different frontends. Therefore, for all j the following holds:
$$y_j = \sum_{i=1}^{N} x_{ij} \ . \tag{6.2}$$

Any backend B_j is composed by m_j machines dedicated to process queries. Whenever a query arrives, it is processed as soon as possible by a server in first-in first-out order. If every server is busy processing other requests, the query will wait in a queue. Since users expect results to be delivered in a short amount of time and queries can not wait indefinitely to be processed [97], each request must be processed within a certain *time budget* τ since its arrival on a frontend. Consequently, a query can be classified according to the time spent in processing it, and the quality of the produced results:

- *successful query* – a server processes the query and successfully terminates within τ milliseconds, generating a complete list of results;

- *approximated query* – a server starts processing the query, but the time budget τ expired before full evaluation. The query processing is early terminated and a partial list of results is generated [33, 61];
- *failed query* – the query spends all its time budget waiting in queue. No server is able to process the query resulting in an empty result list or an error page returned to the user.

In our approach we wish to limit the number of approximated and failed queries as they can negatively impact the user satisfaction. Arguably, the configuration of the system should keep the amount of failed queries close to zero as they might severely hurt the user perception of performance.

Once the result list is generated, the backend B_j sends it back to the frontend F_i which has forwarded the relative query. The underlying communication network introduces latencies which implies that round trip times from D_i to D_j needs to be accounted for. $RTT(i, j)$ is the overall time required by the network to send a request from F_i to B_j, and the response from B_j to F_i. Round trip times negatively impact query processing, as they consume a part of the time budget of forwarded queries[2].

Finally, the number m_j of identical machines hosted by the backend B_j determines V_j, the maximum query volume the backend can sustain, this is, V_j is the maximum number of queries per second that B_j can successfully process before their time budget allowance expires:

$$V_j = \frac{m_j}{\mu} \, , \tag{6.3}$$

where μ is the average query processing time.

The amount of queries y_j received by backend B_j should not exceed its capacity V_j. Whenever this happens the backend becomes overloaded with requests and starts to produce approximated and failed queries. Therefore, for every j, we impose that:

$$y_j \leq V_j \, . \tag{6.4}$$

Moreover, we want any data center to be able to process its locally assigned queries without being overloaded with those forwarded by other frontends. Given so, a remote frontend F_i should limit the number x_{ij} of queries forwarded to B_j by a quantity bounded by B_j's residual capacity, i.e., the difference between the backend capacity and the locally incoming queries, divided by the number of remote sites. In line with [61], for any $i \neq j$ we impose that:

$$x_{ij} \leq \left\lceil \frac{\max(0, V_j - a_j)}{N - 1} \right\rceil \tag{6.5}$$

Energy model In this work we consider the energy consumed by a data center to be accounted exclusively to its correspondent backend. Even if this assumption does not hold in practice (for example, a considerable amount of energy is spent in thermal cooling or inefficient power supplies causing electrical losses), these issues are present regardless of queries being forwarded and do not affect the performance numbers comparisons.

The power P absorbed by a server is a function of its utilization level u [10, 12], which depends on its *busy time* (time spent performing computations). Since B_j's servers are dedicated to process queries, in our scenario a server's busy time accounts for the amount of time needed to process all the queries assigned to the machine. If

[2] $RTT(i, i)$ is assumed to be zero.

Q is the set of queries assigned to a server then the utilization level $u(Q)$ equals:

$$u(Q) = \frac{\sum_{q \in Q} s_q}{\Delta t} \ , \tag{6.6}$$

where s_q is the time required to solve a query q in Q, with $0 \leq s_q \leq \tau$.

Power consumption increases linearly with its utilization and it reaches its peak $\hat{P}$ when $u(Q) = 1$. If a server is idle, i.e. $u(Q) = 0$, it consumes only a small fraction of its peak power $\alpha\hat{P}$, with $0 < \alpha \leq 1/2$ [10]. Therefore, the power consumption of a server as a function of its utilization level $u(Q)$ can be written as:

$$P(Q) = \alpha\hat{P} + (1-\alpha)\hat{P}u(Q) \ . \tag{6.7}$$

At each time slot Δt we consider the energy E_j consumed by the backend B_j to be the sum of the electricity required by its m_j servers. Assuming B_j being composed by homogeneous machines, for all j we have:

$$E_j = \sum_{k=1}^{m_j} P(Q_{jk})\Delta t \ , \tag{6.8}$$

where Q_{jk} is the set of queries processed by the k-th server in B_j. By combining Equations 6.6 and 6.7, we have that:

$$\begin{aligned} E_j &= \alpha m_j \hat{P}\Delta t + (1-\alpha)\hat{P}\sum_{k=1}^{m_j} u(Q_{jk})\Delta t \\ &= \alpha m_j \hat{P}\Delta t + (1-\alpha)\hat{P}\sum_{k=1}^{m_j}\sum_{q \in Q_{jk}} s_q \ . \end{aligned} \tag{6.9}$$

We define Q_j as the full set of queries processed by the j-th backend's machines, i.e., $|Q_j| = y_j$. Finally, the average energy E_j consumed by backend B_j during a time slot Δt can be expressed as a function of the mean processing time μ as:

$$E_j = \alpha m_j \hat{P}\Delta t + (1-\alpha)\hat{P}\mu y_j = I_j + H_j y_j \tag{6.10}$$

The quantity I_j represents the energy consumed by B_j over Δt seconds when all its servers are idle. Instead, the quantity H_j represents the energy consumed to process queries.

Cost model We assume that every data center receives a limited amount of green energy per time slot, denoted as G_j. This green energy is available to the data center because it has its own green power plant, or because of special agreements between the search engine company and the energy provider. If G_j is not sufficient to satisfy the energy consumption E_j the data center needs to buy additional, possibly brown, energy from the market, denoted as M_j.

The number of queries that, on average, can by processed by backend B_j by only exploiting green energy is:

$$g_j = \max\left(0, \frac{G_j - I_j}{H_j}\right) = \max\left(0, \frac{G_j - \alpha m_j \hat{P}\Delta t}{(1-\alpha)\hat{P}\mu}\right) \ . \tag{6.11}$$

We consider that each data center D_j can consume energy available in two different prices per energy units. We denote as p_j^M the unitary cost of market energy at site D_j.

This price varies throughout the day, typically being higher at daytime than during night [91]. Similarly, we define p_j^G to be the price of green energy at data center D_j. We assume the green energy to be more convenient than the market energy [47, 100]. Therefore, we would like to use only green energy to operate the backends and to process search requests, to reduce the search engine operational cost and carbon footprint. However, when G_j units of energy have been used in a given time period, we must resort to buy M_j units of market energy, to power the data center D_j for the remaining of the time slot. Finally, the operational costs C_j of a data center D_j can be expressed as:

$$C_j = \begin{cases} p_j^G E_j & \text{if } E_j \leq G_j \\ p_j^M E_j - (p_j^M - p_j^G) G_j & \text{otherwise} \end{cases} \tag{6.12}$$

Our goal is to minimize the overall operative cost in running a geographically distributed search engine, that is:

$$\min \sum_{j=1}^{N} C_j \ . \tag{6.13}$$

6.3 Problem solution

In this section we propose and discuss a new query forwarding algorithm, called Min Cost Flow (MCF). This algorithm is executed periodically at the frontends of the data centers to decide if an incoming query must be processed locally or forwarded to another data center's backend in order to minimize the overall operational cost of a geographically distributed search engine. During every time interval Δt, we need to know how many queries each data center D_i must process locally or forward to data centers D_j, i.e., we need to compute values for the x_{ij} variables introduced in Section 6.2. In the following, we will show how these can be obtained by solving an instance of the Minimum Cost Flow Problem [15].

Algorithm 5 illustrates the MCF query forwarding algorithm. Firstly, each data center D_i must be able to locally compute the x_{ij}'s values for building a *forwarding table* X_i, such that $X_i[D_j] = x_{ij}$. This table is build every Δt seconds (lines 1–6). In order to generate this table, the algorithm needs an estimate of the query volumes a_i arriving to each data center in the next time slot Δt. The algorithm also requires per-site information related to its maximum sustainable query volume V_i, the available green energy G_i and the energy prices (p_i^M, p_i^G) at that time interval. We assume that data centers exchange messages every Δt seconds containing all those values [61].

At this point, whenever a query q arrives to the frontend F_i, the data center estimates its required processing time s_q (line 8). Next, F_i uses its forwarding table X_i to decide to which sites it could forward a query to (line 9) and the frontend selects in a round-robin fashion a backend B_j from X_i, such that $X_i[D_j] > 0$ (line 13). This decision takes into account the expected round trip time from F_i to B_j. If the sum of the expected query processing time and the expected round trip time is smaller than the time budget τ, then q is forwarded to B_j (line 17). The value $X_i[D_j]$ is decreased by one and j is removed from X_i as soon as $X_i[D_j] = 0$ (lines 14–16). It is possible that no remote backend can process the query within the time budget; in such case the query is processed locally by default (lines 10–11).

Algorithm 5: The MCF query forwarding algorithm.

```
1  every $\Delta t$ seconds do
2      $A = \{a_1, \dots, a_N\}$
3      $V = \{V_1, \dots, V_N\}$
4      $\Gamma = \{G_1, \dots, G_N\}$
5      $\Pi = \{p_1^M, \dots, p_N^M, p_1^G, \dots, p_N^G\}$
6      $X_i$ = GenerateForwardingTable($A$,$V$,$\Gamma$,$\Pi$)

7  forall incoming queries $q \in Q$ do
8      $s_q$ = EstimateProcessingTime($q$)
9      $J = \{D_j : X_i[D_j] > 0 \wedge s_q + RTT(i,j) \leq \tau\}$
10     if $J$ is empty then
11         process $q$ locally
12     else
13         $D_j$ = select $D_j \in J$ in a round robin fashion
14         $X_i[D_j] \leftarrow X_i[D_j] - 1$
15         if $X_i[D_j]$ = 0 then
16             remove $D_j$ from $X_i$
17         forward $q$ to $D_j$
```

Generating a forwarding table The forwarding table X_i used in Algorithm 5 is obtained by solving an instance of the Minimum Cost Flow Problem (MCFP) [15] derived from the model of the distributed search engine discussed in Section 6.2.

The flow starts from some source nodes and every unit of flow needs to reach one or many sink nodes. In our case, queries represent the flow circulating in the network. Nodes are used to represent flow balance equations, i.e., the sum of incoming flow must be equal to the sum of outgoing flow. Every edge is labeled with three values representing the amount of flow passing through the link, its total capacity and cost per unit of flow.

Figure 6.2 illustrates how the operational cost minimization problem with two data centers can be represented as a MCFP instance. Source nodes are the query sources $S_1, \dots, S_N$ described in Section 6.2. These nodes are connected to their correspondent frontends $F_1, \dots, F_N$, using edges with infinite capacity and zero cost. Each source node S_i originates a_i units of flow (i.e., queries). It is not known, a priori, how many queries will arrive on frontend F_i during a particular time slot Δt. For this reason, a_i is estimated based on previous query arrivals as detailed in Section 6.4.

Each frontend F_i is connected to every backend B_j. The arcs among frontends

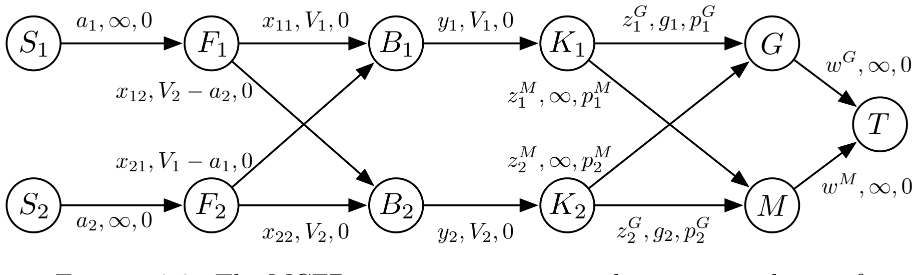


Figure 6.2: The MCFP instance to minimize the operational cost of a geographically distributed search engine.

and backends have no cost. However they have different capacities. Edges (F_i, B_i) have capacity V_i, i.e., the maximum volume of queries per second sustainable by backend B_i (see Equation 6.4). Instead, edges (F_i, B_j) with $i \neq j$ have capacity $\lceil \max(0, V_j - a_j)/(N-1) \rceil$, as per Equation 6.5. The flows on the (F_i, B_i) edges represent the values for the x_{ij} variables defined in our model.

An artificial node K_j is introduced for every backend B_j, connected through an edge with zero cost and capacity V_j. These nodes enforce the constraint $y_j \leq V_j$ for every backend B_j (see Equation 6.4). The nodes K_j are connected to two more nodes, namely *Green* (G) and *Market* (M). The first node models the query processing by using green energy, while the second node by using market energy. Each K_j node is connected to the *Market* node through edges with cost p_j^M and infinite capacity. Moreover, each K_j node is connected to the *Green* nodes by edges with cost p_i^G but limited capacity g_i, i.e., the number of queries that B_i can process by using just green energy (see Equation. 6.11). The flows z_j^G and z_j^M are just an artifact included to satisfy the flow balance equations $y_j = z_j^G + z_j^M$.

Finally, the G and M nodes are connected to the sink node T. The edges ending in T have zero cost and infinite capacity and the flows w_G and w_M are an artificial mean to guarantee that the flow entering in T is equal to the flow leaving the sources S_i, i.e., $\sum_{i=1}^{N} a_i = w_G + w_M$.

Note that every edge but (C_i, G) and (C_i, M) has zero cost. Therefore, to minimize the flow cost indirectly minimizes the cost function defined by Equation 6.13.

MCFP instances can be solved in polynomial time by using linear programming tools. More importantly, while MCF runs at every query arrival, the GENERATEFORWARDINGTABLE function needs to be executed less frequently (every Δt seconds).

Estimating the query processing time There are several approaches to estimate the processing time of a query *before* executing it in the search engine backend. Documents relevant to a query are retrieved from the inverted index, by exhaustively traversing the posting lists relative to the query terms. In this case, processing time relates to the posting list lengths of the query terms [85]. Dynamic pruning techniques, such as MaxScore [105], can be deployed to speed up query processing. These techniques skip over portions of the posting lists by avoiding the evaluation of non relevant documents. When dynamic pruning is applied, queries with the same number of terms and similar posting list lengths can have widely different processing times and query efficiency predictors can be used [75].

Algorithm 5 is agnostic to the particular ESTIMATEPROCESSINGTIME() implementation. In the experimental section, we assume that this function uses an oracle which is employed in both the baseline and new algorithm.

6.4 Experimental setup

In this section, we determine empirically the potential of the proposed model and algorithm for reducing the operational cost of a distributed search engine without negatively impacting on its efficiency. In particular, we address three research questions, as follows:

- RQ1: What is the impact of the proposed approach and the baselines on the quality of service of the search engine?
- RQ2: Does our approach diminish the data centers' carbon footprint?

- RQ3: Do our MCFP-based model and the query forwarding MCF algorithm achieve energy cost savings comparable with reasonable baselines?

To answer questions 1, we measure the number of approximated and failed queries produced by the search engine. To answer question 2, instead, we need to evaluate the goodness of system in exploiting green energy. In fact, when the search engine misses the opportunity to consume green energy, it turns to the energy market – buying possibly brown energy and increasing its carbon footprint. For this reason we measure the system *green energy efficiency*, defined as the ratio between the amount of used green energy and the amount of green energy globally available to the search engine. Finally, to answer research question 3, during our simulations we measure the overall electricity expenditure of the search engine.

In the remainder of this section we define the experimental setup to address our research questions covering the baselines, the data centers, the data, the energy prices and the workload estimates.

Baselines We will compare these results against two baselines. The first baseline, called NoForwarding, represents a standard multi-center web search engine. It does not perform any query forwarding: the queries received by a frontend are processed in the local backend.

The second baseline forwards the queries following the approach in [61]. According to this technique, the probability of forwarding a query towards a particular data center is proportional to the amount of queries processable by the remote site and evenly shared among sites with a higher energy prices. Since queries are forwarded using a probability distribution, we here refer to this approach as PROB. The technique works as follows. At each query arrival, the data center D_i estimates the workloads (i.e., incoming queries) of each data center. Then, D_i looks at which data center is underloaded, i.e., which site has the opportunity to process forwarded queries. If these other data centers have a lower energy price, D_i simulates to redistribute its own workload towards these sites. However, D_i conservatively assumes that other data centers will try to do the same. Therefore, D_i equally divides such forwarding opportunity with its "competitors". The remaining queries are simulated to be processed locally. At the end of the simulation, the data center D_i has computed how many queries x_{ij} it would forward to data center D_j. A probability distribution is generated accordingly to these values and the incoming query is forwarded to a data center following such distribution.

We do not consider here the approach proposed in [101], as it does not perform any query forwarding in the case of fully replicated indices, therefore providing no benefits in our scenario.

Search data centers Our experiments simulate a real distributed web search engine with six data centers: DC-A, DC-B, DC-C, DC-D, DC-E and DC-F. We assume that these data centers are located in the capital cities of six different countries. In order to not disclose sensitive information, the countries are not revealed. We approximate network latencies between frontends and backends by considering the speed of light on copper wire (200,000 km/s) and the bird-fly distances between the relative cities [22, 61].

We assume that the data centers' backends contain identical servers. In this work, we experiment with two different kinds of server, i.e., we variate the α parameter defined in Section 6.2. The first server type represents normal servers which consume

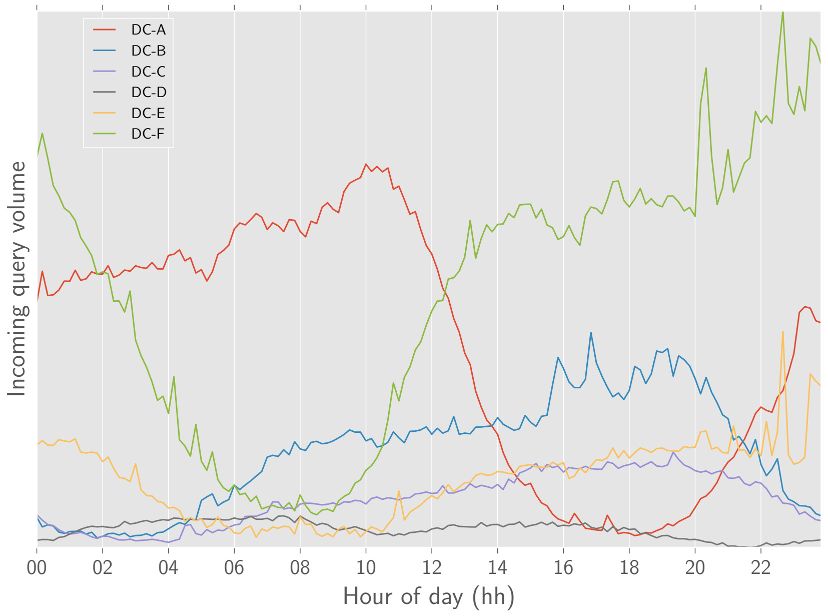


FIGURE 6.3: Query workload for six different Yahoo frontends during 24-hours.

half of their peak power when idle (i.e., $\alpha = 0.5$). The other type represents next-generation servers, which are more energy-proportional, with $\alpha = 0.25$ [10].

The number of machines in a backend is determined by taking the reciprocal of the backend capacity in Equation 6.3. The maximum sustainable query volume, V_i, is set to be the 99.95th-percentile of the observed query loads, taken every second over the logs.

Data We use real-world queries and arrival times sampled – in the order of tens of millions of points – from six different frontends of the Yahoo web search engine, spanning one week. The first day of the log (see Figure 6.3) is used to tune our approach and the baselines. The remaining days are used in the simulation, i.e., queries are sent to the simulated data centers following the same load and arrival times reported in the log.

The processing times, on the other hand, are randomly sampled from an empirical distribution. This distribution is obtained by processing one million of unique queries using Terrier [76]. The platform is hosted on a dedicated Ubuntu 14.04 server, equipped with 32GB RAM and an Intel i7-4770K processor. Queries are taken from the MSN 2006 log [83], while documents relevant to such queries are retrieved from the TREC ClueWeb09 collection (cat. B) [67]. The corpus is indexed removing stopwords and stemming terms. The Elias-Fano schema is used for compression [107], and the resulting inverted index is kept in main memory. At retrieval time, dynamic pruning is applied by using the MaxScore algorithm [105]. The experimental mean processing time is 100 ms and its 99th-percentile is 589 ms. During our simulations, the query time budget τ is fixed to the reasonable value of 500 ms. This threshold has been chosen to avoid a drop ratio greater than 0.5%. We note that this value can be re-scaled if we consider shards with different sizes, for instance, when each machine index stores a lower amount of web pages.

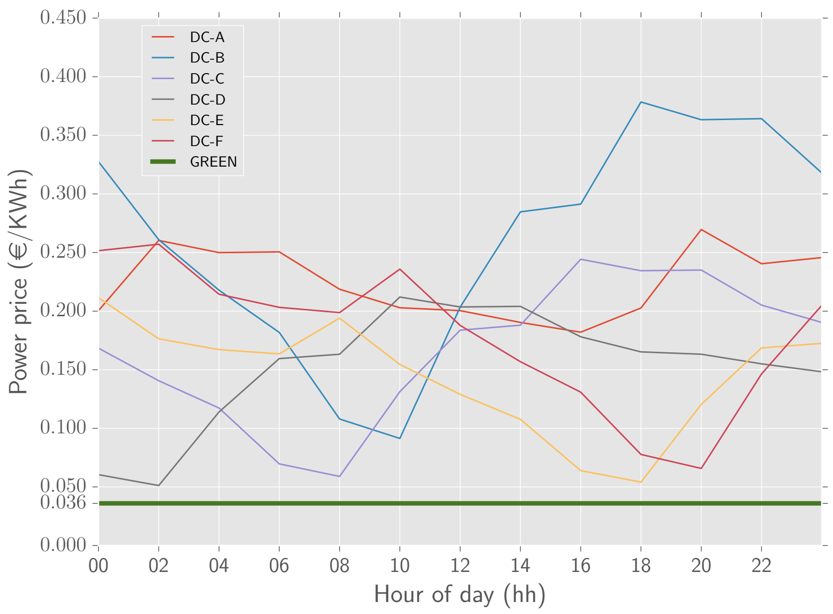


FIGURE 6.4: Market electricity prices and green energy price.

Electricity prices We need to set the price of green and market energy available in each data center, along with the amount of green energy available at each location.

Market electricity price varies by country and by hour of the day. To simulate this behavior, we generate a market electricity cost configuration in each data center. This configuration follows the price fluctuations observed over one week on the United Kingdom day-ahead market [89]. Prices are modified to reflect the cost of energy in the countries where the data centers are placed [109]. Also, prices are shifted accordingly to the timezone of the corresponding data center country. One day of the resulting electricity costs is shown in Figure 6.4. All times are normalized to the Greenwich Mean Time, i.e., GMT+0.

We assume that data centers have their own green power plants. Considering construction and maintenance costs, these produce energy for 0.036 €/KWh, which is cheaper than market energy [100]. As stated in Section 6.2, we assume that each data center D_i receives a fixed amount G_i of green energy at every time slot Δt. This quantity covers a fraction of the maximum amount of energy that D_i can possibly consume in Δt seconds. The maximum energy consumption of a data center is given by the product of the number m_i of its servers times their peak power consumption $\hat{P}$ times the time slot length Δt. Therefore, at each time slot, a datacenter D_i receives $G_i = \gamma m_i \hat{P} \Delta t$ KWh of green energy. We denote by γ the *green energy availability ratio* and range this value from 0 to 1 in the experiments. Green energy availability can fluctuate over the day. For instance, solar and wind energy production is susceptible to weather conditions. While our experiments do not consider this aspect, MCF can deal with variable green energy availability.

Workload estimate Both PROB and MCF algorithms need to accurately estimate the workloads of data centers $a_1, \ldots, a_n$ in every time slot, to decide where to forward queries. Following [61] we assume that data centers exchange messages about their current workload every second (i.e., $\Delta t = 1s$). This also corresponds to the execution frequency of the GENERATEFORWARDINGTABLE function (see Algorithm 5). In the succeeding time slot, each data center estimates the workloads of remote sites by

Table 6.1: Time windows size (in seconds) for estimating the incoming query workloads of different data centers.

	$\alpha = 0.25$		$\alpha = 0.5$	
γ	PROB	MCF	PROB	MCF
0.0	200	40	200	40
0.1	200	40	190	40
0.2	200	40	200	40
0.3	200	40	200	40
0.4	200	40	190	40
0.5	200	1	210	40
0.6	200	1	190	40
0.7	190	10	200	1
0.8	200	1	200	10
0.9	200	1	200	1
1.0	200	10	200	10

using their recent history. Conservatively, a future workload is approximated to the maximum query volume observed over a time window for that remote data center [61]. In practice this implies that we need to select a time window size such that the workload estimates are as accurate as possible. As a working example consider a data center DC-A that is highly loaded. If the window size is too small and the query load briefly drops, we incur in the risk of forwarding too many queries to DC-A erroneously assuming that it is now lightly loaded. This situation will result in a high number of failed queries. On the other hand, if the time window is too large and DC-A is actually becoming lightly loaded, we will mistakenly produce low forwarding rates assuming that DC-A is still highly loaded. This implies that the algorithm will effectively miss the opportunity to save costs. Therefore, we use the first day of query log to determine a reasonable time window size. We set this size to the minimum possible value such that the failed query rate remains below the 0.5% threshold, as in [61]. The ideal time windows are reported in Table 6.1 and consistently used for the experiments. Note that MCF needs smaller time windows than PROB to estimate the incoming workloads.

6.5 Results

In this section, we report on a comprehensive evaluation of MCF and PROB. The reported results are presented as percentage improvements over the NoForwarding baseline. We compare consistently the NoForwarding, PROB and MCF algorithms with varying configurations, namely the idle server power consumption fraction α and the data center green energy availability γ. We assume $\alpha = 0.5$ for current servers and $\alpha = 0.25$ for next-generation servers, while we vary γ from 0 (data centers completely powered by market energy) to 1 (data centers completely powered by green energy).

We aim to determine if our proposed MCF algorithm allows to markedly reduce market power consumption and operational costs with latency comparable to that achieved by the baselines. In particular, we firstly discuss the impact of the MCF approach on the quality of service of the system, measured in terms of the amount of approximated and failed queries. Then we evaluate the eco-friendliness of our algorithm with respect to the baselines, measured in terms of their green energy efficiency. Finally, we address our key research question concerning the cost savings MCF can obtain. The outcome of the experiments is summarized in Table 6.2.

Quality of service The two first main columns of Table 6.2 report the values concerning the degradation in effectiveness of the different approaches, when queries are being processed in a distributed fashion across different data centers.

Results state that our proposed approach does not negatively impact the overall service quality of the multi-site web search engine. Across all the tested green availability ratios γ, the MCF algorithm reduces the number of approximated queries. Moreover, thanks to query forwarding, approximated queries can be reduced from 1% to 11% percent with respect to NoForwarding, when currently existing servers (i.e., $\alpha = 0.5$) are utilized. Similar values are observable when the data center is equipped with more energy-proportional servers ($\alpha = 0.25$).

Table 6.2 also shows that MCF achieves the best absolute result in reducing the approximated queries with respect to the PROB baseline when a large fraction of the data centers is powered by green energy. However, while the PROB baseline maintains an almost constant decrease of ∼11% of approximated queries across all green energy availability configurations, MCF does not forward many queries for high green energy availability values, since a large quantity of cheap green energy is available locally at each data center. Consequently, data centers may incur in local overload situations that force them to early terminate some queries.

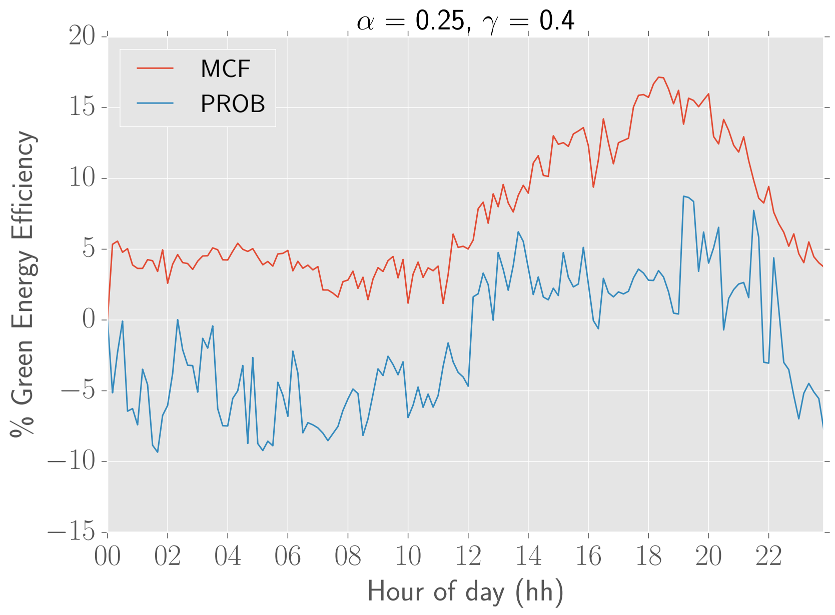
We do not report here the number of failed queries since in all configurations this value is always below the 0.5% threshold imposed in Section 6.4. Similarly, we do not report here the query response times, as the query time budget τ is fixed to 500 ms.

Green energy efficiency As seen in Section 6.4, green energy efficiency gives us a measure of the search engine eco-friendliness. Table 6.2 shows that query forwarding plays a limited role in improving the green energy efficiency of a multi-site search engine built of current technology servers when the amount of green energy is limited ($\gamma \leq 0.5$). This effect happens because half of the energy consumed by the data centers is used just to keep the servers operative and idle, without processing any query. Therefore, there is no opportunity for processing requests using green energy when $\gamma \leq 0.5$. Similarly, MCF and PROB behave like NoForwarding also when green energy is abundant, i.e. $\gamma \geq 0.8$; in this case, in fact, a query is processed using green energy whether it is forwarded or not. With current servers, MCF shows eco-friendliness when $0.5 < \gamma < 0.8$, up to 1.66% when $\gamma = 0.6$. On the other hand, PROB proves to be less green energy efficient than NoForwarding. In fact, PROB is unaware of green energy availability and forwards queries where market energy is less expensive. While doing this, PROB misses the opportunity to exploit locally available green energy. Furthermore, it will not forward queries to data centers where market energy is costly but green energy is available.

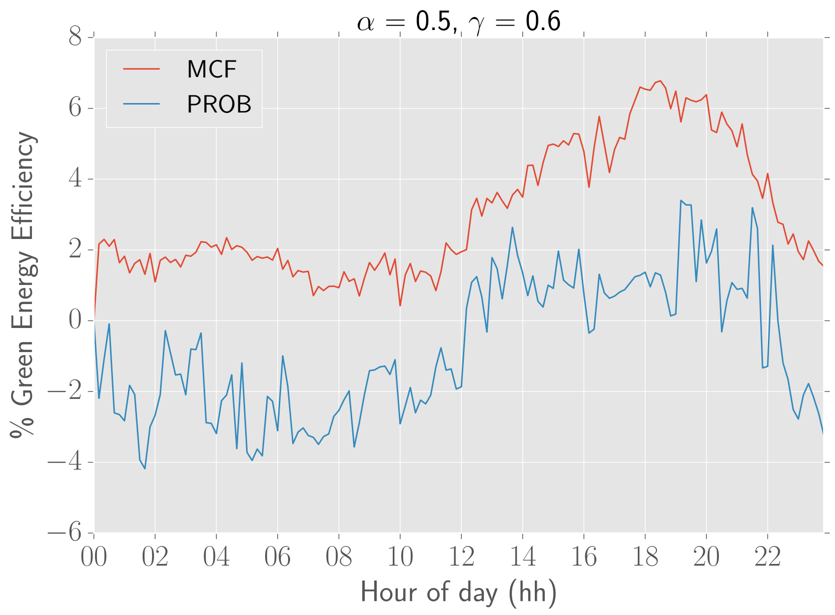
Green energy efficiency improves when employing more energy-proportional servers. If $\alpha = 0.25$, MCF becomes more effective in exploiting green energy, improving over a larger range of green energy availability $0.3 < \gamma < 0.8$, up to 5.37% when $\gamma = 0.5$. Larger effects can be noticed on a smaller timescale, as highlighted in Figure 6.5. Instead, the PROB baseline proves to be inadequate in reducing the carbon footprint of a search engine, even when energy-proportional servers are employed.

Table 6.2: Percentage of (a) Approximated queries, (b) green energy efficiency improvements, and (c) cost savings with respect to NoForwarding. The best results are reported in **bold**.

	(a) Approximated queries				(b) Green energy efficiency				(c) Cost			
	$\alpha = 0.25$		$\alpha = 0.5$		$\alpha = 0.25$		$\alpha = 0.5$		$\alpha = 0.25$		$\alpha = 0.5$	
γ	PROB	MCF	PROB	MCF	PROB	MCF	PROB	MCF	PROB	MCF	PROB	MCF
0.0	-11.15	-10.70	-11.16	-10.84	0.00	0.00	0.00	0.00	-9.86	-9.45	-4.71	-4.50
0.1	-10.95	-10.77	-11.08	-10.85	0.00	0.00	0.00	0.00	-11.96	-11.55	-5.39	-5.17
0.2	-11.20	-10.89	-11.03	-10.80	0.00	0.00	0.00	0.00	-15.44	-14.82	-6.30	-6.07
0.3	-11.03	-10.85	-10.82	-10.60	-2.58	0.26	0.00	0.00	-16.84	-19.52	-7.64	-7.34
0.4	-11.23	-11.46	-11.35	-10.95	-4.07	3.93	0.00	0.00	-9.02	**-25.51**	-9.75	-9.30
0.5	-11.16	-10.08	-11.11	-10.84	-2.19	**5.37**	0.00	0.00	-1.35	-21.73	-13.21	-12.67
0.6	-10.98	-7.77	-11.03	-11.21	0.02	1.14	-1.76	**1.66**	-1.23	-5.75	-5.52	**-15.79**
0.7	-11.19	**-11.73**	-10.98	-8.48	0.06	0.10	-0.35	1.22	-0.49	-0.66	-0.37	-6.16
0.8	-11.07	-3.21	-10.88	**-11.37**	0.00	0.00	0.03	0.04	-0.04	-0.17	-0.21	-0.28
0.9	-10.99	0.85	-11.16	-1.25	0.00	0.00	0.00	0.00	-0.04	-0.20	-0.01	-0.08
1.0	-11.07	-6.83	-11.20	-7.01	0.00	0.00	0.00	0.00	-0.03	-0.09	-0.00	-0.04



(a)



(b)

FIGURE 6.5: Percentage of green energy efficiency improvements w.r.t. NoForwarding on a daily scale, over the first day of the test query log, for (a) $\alpha = 0.25, \gamma = 0.4$ and (b) $\alpha = 0.5, \gamma = 0.6$.

Cost savings Both MCF and PROB algorithms can help reduce the energy operational cost of a multi-site search engine, as shown in Table 6.2. When current servers are used, MCF and PROB similarly behave until the green energy availability $\gamma \leq 0.5$. Indeed, half of the energy available has to be consumed just to keep them operative. However, in order to be able to process queries, data centers need to buy additional energy from the energy market leaving no opportunity for MCF to exploit green energy for query processing. In any case, even when $\gamma \leq 0.5$, both PROB and MCF forward queries to sites with cheaper market energy, successfully reducing the operational costs of the whole search engine. As highlighted in Figure 6.6, when green energy availability is $0.5 < \gamma < 0.8$, MCF reduces the energy operational cost up to almost 16% w.r.t NoForwarding. Such savings are even higher if we look at daily variations, as shown in Figure 6.7.

Conversely, PROB reduces the operational cost by only less than 6% when $0.5 < \gamma < 0.8$. This effect happens again because PROB is not able to use the information about the green energy availability, while MCF is able to exploit this knowledge. When $\gamma \geq 0.8$, little differences can be see with NoForwarding as green energy becomes highly available at every sites, and query forwarding does not make any difference. If we use more energy-proportional servers, larger benefits are observable over a wider range of green energy availability. MCF reduces the energy expenditure of NoForwarding by more than 25% when just 40% of green energy is available. Again, these benefits are even larger when considering smaller time scales, as per Figure 6.7. Under the same configuration, PROB can only save less than 10%.

Finally, it is important to highlight that MCF obtains the best results when $0.3 < \gamma < 0.8$. This reinforces the importance of our results, as data centers would probably work with limited amounts of green energy due to its susceptibility to external variables such as the weather conditions.

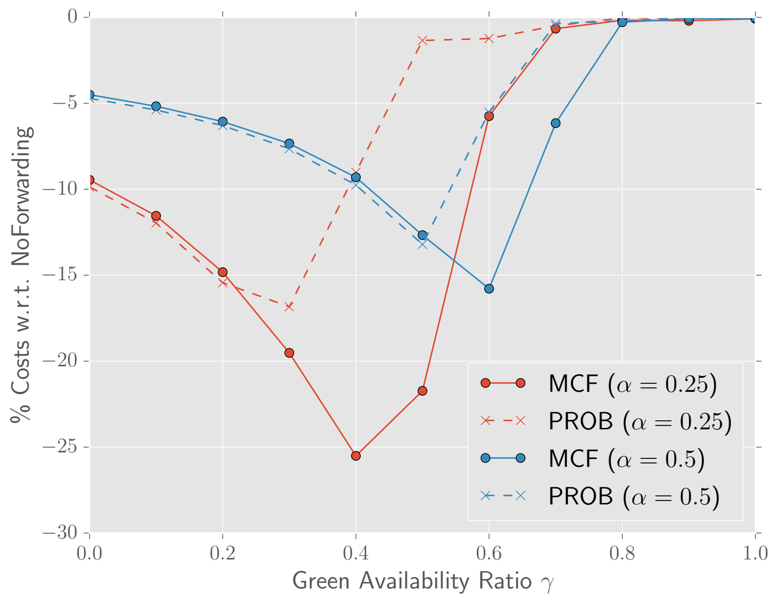
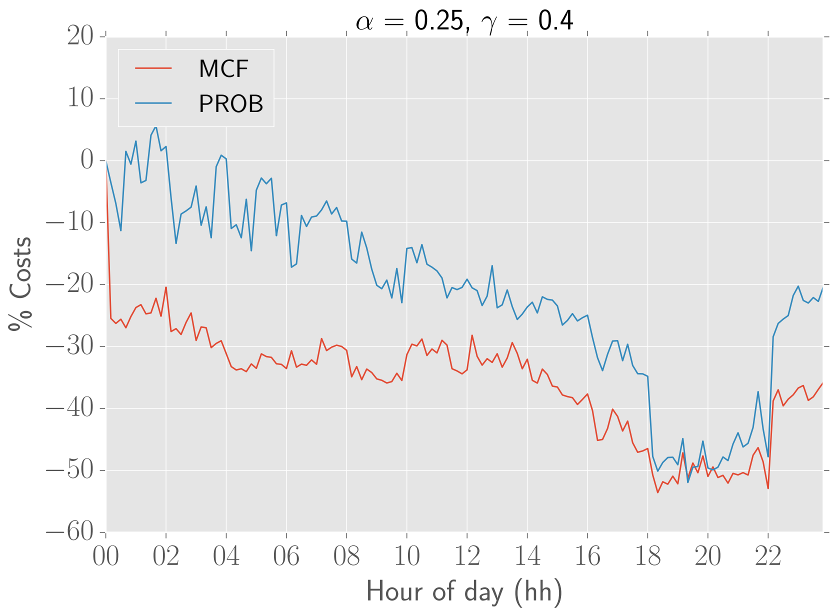
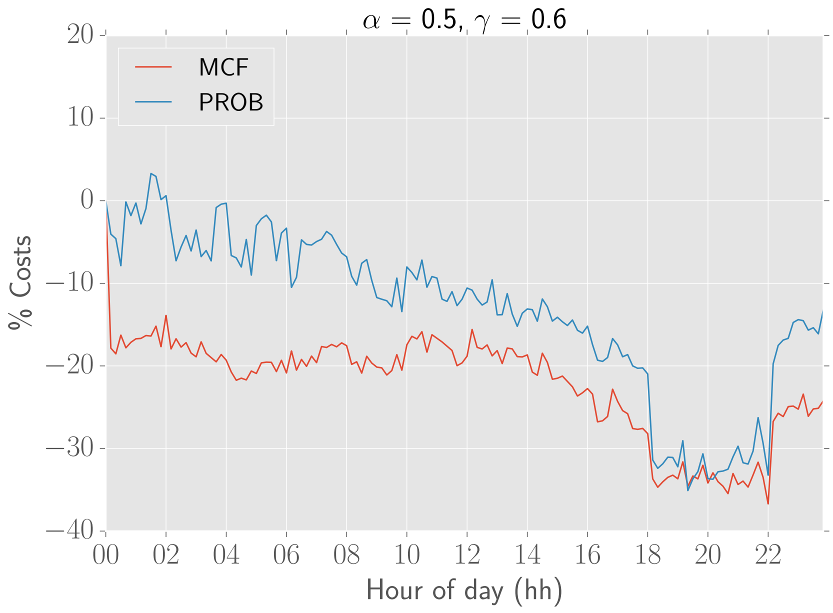


FIGURE 6.6: Percentage of cost savings w.r.t. NoForwarding, for different values of α and γ.



(a)



(b)

FIGURE 6.7: Percentage of cost savings w.r.t. NoForwarding on a daily scale, over the first day of the test query log for (a) $\alpha = 0.25, \gamma = 0.4$ and (b) $\alpha = 0.5, \gamma = 0.6$.

6.6 Discussion

In this chapter we proposed a novel model of a multi-center web search engine that characterizes the frontend and backend components of the system, together with their processing capacity, to represent the energy operational cost of the search engine in presence of green energy resources. Using this model, we proposed MCF, a novel query forwarding algorithm based on a Minimum Cost Flow Problem formulation. MCF is able to leverage the different pricing and availability of brown and green energy to reduce the energy expenditure of search systems. At the same time, MCF effectively takes into account the various backend processing capacities to maintain an acceptable service quality of the system.

We simulated the proposed algorithm using real query workloads obtained from six frontends of a live search engine and realistic electric price data. We compared our results with two baselines: NoForwarding, which represents a standard multi-site search engine; and the state-of-the-art PROB algorithm, which greedily forwards queries to the cheapest site following a estimated probabilistic distribution. We showed that with current technology servers, query forwarding plays a limited role in reducing the carbon footprint of search engines, although when a data center is equipped with next generation hardware query forwarding is a promising and effective technique to exploit green energy usage. This further reinforces the need for energy-proportional hardware in web search engine data centers. Finally, we illustrated how MCF performs better than PROB in achieving cost savings, obtaining energy expenditure reductions that range from $\sim$15% to $\sim$25% with respect to the NoForwarding configuration. These savings are possible when limited quantity of green energy are available at the different remote sites, stating the importance of the MCF algorithm since green energy is typically attainable only in limited quantities due to external variables like weather conditions among others.

For future work, we will consider more complex scenarios where different green power sources like solar and wind are available, and they vary dynamically during a day at the different sites as well as the development of more complex query workload forecast predictors. Also, it will be interesting to study the interactions among query forwarding techniques and energy saving approaches based on DVFS technologies [24, 26, 37, 71]. For instance, PESOS tries to process queries no faster than user expectation to save energy by exploiting low CPU frequencies. On the contrary, query forwarding techniques may require that query are always processed at maximum CPU frequency, to compensate the network delays introduced by the workload offload. Whether these two aspects are reconcilable is left for future investigations.

Chapter 7

Conclusions and Future Directions

In this thesis, we illustrated the energy-related challenges incurred by Web search engines. In fact, these systems consume an huge amount of electricity to operate their data centers. Such consumption are very expensive, and negatively impact on the environment due to the carbon dioxide emitted to produce electricity.

In Chapter 1 and 2 we identified and discussed three levels of a search engine architecture on which it is possible to intervene for reducing its energy expenditure and carbon footprint. At the intra-server level, energy management approaches aim at reducing the energy expenditure of the hardware components of single servers within a Web search engine. Instead, energy management approaches operating at the intra-data center level coordinate the operations of computational resources within a single data center to reduce its energy expenditure. Finally, at the inter-data center level the operations of multiple data centers can be orchestrated to reduce the overall energy expenditure or carbon footprint of a multi-center search engine.

In this thesis, we mainly focused on improving the energy efficiency of the query processing subsystem of a Web search engine. In particular, we studied the problem at the intra-server and inter-data center level. Regarding the intra-server level, in Chapter 3 we showed that opportunities exist for a search engine to reduce the energy consumption of its search servers by exploiting CPU frequency scaling mechanisms. Therefore, in Chapter 4 we proposed to delegate the control of the CPU frequencies from the operating system to the query processing application. In this way, the operating frequency of the CPU cores can be managed by the search engine which can exploit information such as the search application utilization and load. This knowledge permits to better control frequency scaling with respect to the operating system. Indeed, experimental results show that the approaches we presented in Chapter 4 allow reducing up to 24% a server power consumption, with only limited (but uncontrollable) drawbacks in the quality of search results with respect to a system running at maximum CPU frequency.

In Chapter 5, we kept investigating in how to reduce energy consumption at the intra-server level. Stemming from the observations of Chapter 3, in this chapter we advised that Web search engine should not process queries faster than user expectations, to reduce CPU energy consumption. In fact, users can hardly notice response times that are faster than their expectations. Consequently, we proposed the Predictive Energy Saving Online Scheduling (PESOS) algorithm, to select the most appropriate CPU frequency to process a query by its deadline, on a per-core basis. PESOS can reduce the CPU energy consumption of a query processing server from 24% up to 48% when compared to a high performance system running at maximum CPU core frequency. Additionally, PESOS outperforms the best approach presented in Chapter 4 with a 20% energy saving, while the competitor requires a fine parameter tuning and it may incur in uncontrollable latency violations.

Finally, in Chapter 6 we moved our attention on the inter-data center level and we focused on reducing the carbon footprint and energy expenditure of multi-site search engines. To this end, we tackled the problem of targeting the usage of green energy available at different, geographically distant data centers. Therefore, we proposed a new query forwarding algorithm that exploits both the green energy sources available at different data centers and the differences in market energy prices. The proposed solution maintains a high query throughput, while reducing by up to 25% the energy operational costs of multi-center search engines.

In a future work, we will study how to further mitigate the energy-related problems at the intra-data center level. Indeed, se-cons, se-load, and PESOS can be deployed in replicated and distributed search architecture since they are completely decentralized. Nevertheless, it would still be interesting to evaluate the performance of our approaches in a distributed context, where each search server holds a different index shard and query processing times depend on the last server to complete a query's processing. Since a server energy consumption is linear in its utilization, it may also worth investigating the role of index partitioning strategies on the energy efficiency of search clusters. In fact, different partitioning strategies cause different load distributions on the servers, possibly influencing their energy consumption. Furthermore, it is worth studying the interactions among energy saving approaches based on DVFS technologies (e.g., [24, 26, 37, 71]) and query forwarding techniques (e.g., [13, 61]). Indeed, approaches such as PESOS aim at satisfying user expectations while processing queries as slow as possible, exploiting low CPU frequencies to save energy. Conversely, query forwarding techniques may require that query are always processed at maximum CPU frequency, to compensate the network delays introduced by the workload offload.

Finally, we envision that prediction models will play an important role in improving the energy efficiency of Web search engines. For instance, in PESOS we leveraged the query efficiency predictors originally proposed in [75] to predict query processing times. Subsequent works have proposed more accurate query efficiency predictors [59, 62, 112], and we plan to investigate whether better prediction models can improve the energy savings provided by PESOS. Instead, in Chapter 6 we used query workload predictions to guide our query forwarding algorithm. Our simplistic yet effective approach estimates that the number of queries received by a data center at a given instant is similar to the volume of queries received in precedent instants. However, more complex models could take into account other aspects, such as the seasonality of query workloads [98]. Therefore, a future work will study whether taking into account such features can improve our results. Similarly, weather forecast predictors can play a role in devising more effective query forwarding algorithms. Indeed, in this thesis we assumed that green energy is always available. However, the availability of solar and wind energy, for instance, vary dynamically during a day and across the planet. Therefore, weather forecast predictions can help to decide where to shift query processing to further reduce the energy expenditure and carbon footprint of multi-site search engines.

Appendix A

On the CPU Power Consumption

Barroso, Clidaras, and Hölzle reports that "the CPU is the dominant energy consumer in servers, using two thirds of the energy at peak utilization and about 40% when (active) idle." [10, pages 78-79]. In particular, the authors referred to the power usage of Google servers as the compute load varies from idle to full activity levels. To confirm this finding, we conducted the following experiments to measure the CPU power consumption with respect to the overall power consumption of a server, measured through the wall outlet. In particular, we were interested in studying the power consumption of the server used for experiments in Chapter 4 and 5. This server is equipped with 32 GB RAM and an Intel i7-4770K CPU, a member of the Haswell product family. This CPU has eight logical cores that are mapped into four physical cores, thanks to hyperthreading.

To study the power consumption of our server, we implemented a Java program computing the square root of random integers, stored in main memory, to stress the CPU (a similar approach is used in the `stress` utility [108]). Then, we used this benchmark to keep all the server's cores utilized at given level for 10 minutes. At the same time, we measured the energy consumption of the server from the wall outlet, using an Alciom PowerSpy2 power meter [3]. Meanwhile, we measured the energy consumption on the CPU using the Mammut library [30]. We repeated this experiment varying the CPU utilization from 0% to 100%.

Regarding the server used in our experiments, we found that the CPU is responsible for more than 34% of the whole server power consumption, when active idle (see Figure A.1). When the system is at peak utilization, instead, more than 50% of the server energy consumption can be accounted to the CPU. This investigation empirically confirms what reported in [10], although the exact percentage differs, since our server is probably different from the one used in [10]. The experiment code is available at `https://goo.gl/6lkT8K`.

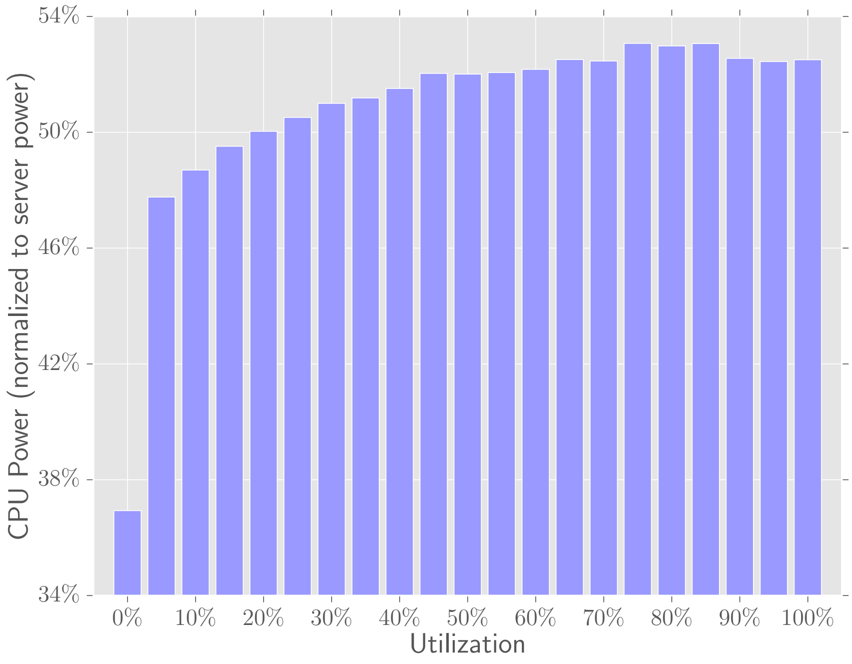


FIGURE A.1: CPU power usage of the server as the compute load varies from (active) idle to full usage.

Appendix B

On the Effectiveness of C-states and P-states

In Chapter 4 and 5 we proposed several approaches to reduce the CPU energy consumption of a search servers by managing its CPU's P-states, i.e., by varying its CPU cores' operating frequencies. In both chapters, we exercise all the CPU cores of the search server to process queries, regardless the incoming query workload. However, a reader may argue that a more sensible strategy would exploit periods of low load to leverage core C-states, i.e., inactive, power saving states.

For instance, a search server could start with just one active core, in low frequency, while the other cores are kept inactive. As the query load increases, the query processing system would ramp up the active core frequency. If a single core cannot handle the load, the system would activate a second core at low operating frequency. This would ramp up as the query load increases, until a third core is needed, and so on. In fact, it would be expected that to keep few cores active (at high frequencies) and the rest of them inactive should save more energy than keeping all of them active at low frequencies.

However, Meisner et al. report that CPU idle states provide little energy savings in web search engines [80]. Moreover, Lo et al. evaluate a solution similar to the one proposed in the previous example [71]. The authors find that keeping few cores active at high frequency consumes more power than keeping all the cores active at lower frequency.

Nevertheless, we decided to experimentally investigate the effectiveness of C-states with respect to P-states. To this end, we used the Java benchmark introduced in Appendix A to investigate the impact of workload balancing, P-states and C-states across cores on the energy consumption of the CPU. We use our application to exercise two of the available CPU's cores available on our server, which is equipped with 32 GB RAM and an Intel i7-4770K CPU. The active cores were allowed to independently select their best operational frequency depending on the respective loads, by using the `powersave` governor. Instead, the other cores were kept pinned to inactive C-states.

Firstly, we executed an instance of our program on a single core for 10 minutes, keeping its utilization at 100%, while the other core was in idle, power saving, C-state, and then we simulated a scenario where the workload is evenly split across the two cores, i.e., we run two instances of the program for 10 minutes keeping the cores' utilization at 50%. Using the Mammut library [30], in the first case we measured a CPU energy consumption of 13.4 KJ versus 10.9 KJ in the second case.

Secondly, we simulated the arrival of an additional workload (e.g., starting a new test process), for a total utilization of 120%, and we compared again the two solutions: keeping a core fully utilized and another one lowly utilized, versus evenly balancing the new total workload onto the two cores. To this end, we first ran two instances of the program for 10 minutes on two cores, keeping one core at 100% utilization and the other one at 20% utilization. Then, we re-ran the instances on the two cores for 10 minutes, while keeping the utilization at 60% on both cores. In the first case, we measured a consumption of 15.5 KJ, while in the second case it was just 13.1 KJ.

These experimental results provide additional evidence that keeping cores active and lowly utilized is better than keeping few cores active but highly utilized and the remaining cores idle. Consequently, in this thesis we mostly focused on managing P-states to improve the energy efficiency of a query processing system.

The source code of these experiment is available at `https://goo.gl/zLHhw5`.

Appendix C

On the Reliability of RAPL Measurements

The ability to accurately measure the CPU energy consumption is fundamental to evaluate the effectiveness of CPU energy saving approaches. To this end, it is possible to measure the overall energy consumption of a server from the wall outlet. However, such measurements are inherently noisy, and they are highly dependent on external factors such as the power grid status. Moreover, they report the overall energy consumption of the server rather than the energy consumed by its CPU.

Recently, Intel introduced the Running Average Power Limit (RAPL) technology. Via the RAPL interface, it is possible to measure the energy consumed by an Intel CPU. These measurements are not synthetic: indeed, since the introduction of the Haswell processor architecture[1], the RAPL component performs actual measurements of the CPU energy consumption. The RAPL interface has been used in other works to measure the energy consumption of CPUs [29, 31], and Hackenberg et al. showed the reliability of such measurements [52]. In particular, they find that the AC power P_{AC} absorbed by a server can be approximated with $R^2 > 0.9998$ with an almost linear fit of the RAPL power.

To confirm the findings in [52], we performed the following experiment using the server and benchmark introduced in Appendix A. We run our benchmark to keep all the server's cores utilized at given level for 10 minutes. At the same time, we measured the energy consumption of the server from the wall outlet, using an Alciom PowerSpy2 power meter [3]. Meanwhile, we measured the energy consumption on the CPU using the Mammut library [30]. We repeated this experiment varying the utilization from 0% to 100%. The results in Figure C.1 confirm an almost linear correlation of the energy consumption reported by the RAPL interface for the CPU (on the x axis) with that reported by the power meter for the whole server (on the y axis). The experiment code is available at `https://goo.gl/6lkT8K`.

[1]Experiments in Chapter 4 and 5 were conducted on a server equipped with an Intel i7-4770K CPU, a member of the Haswell product family.

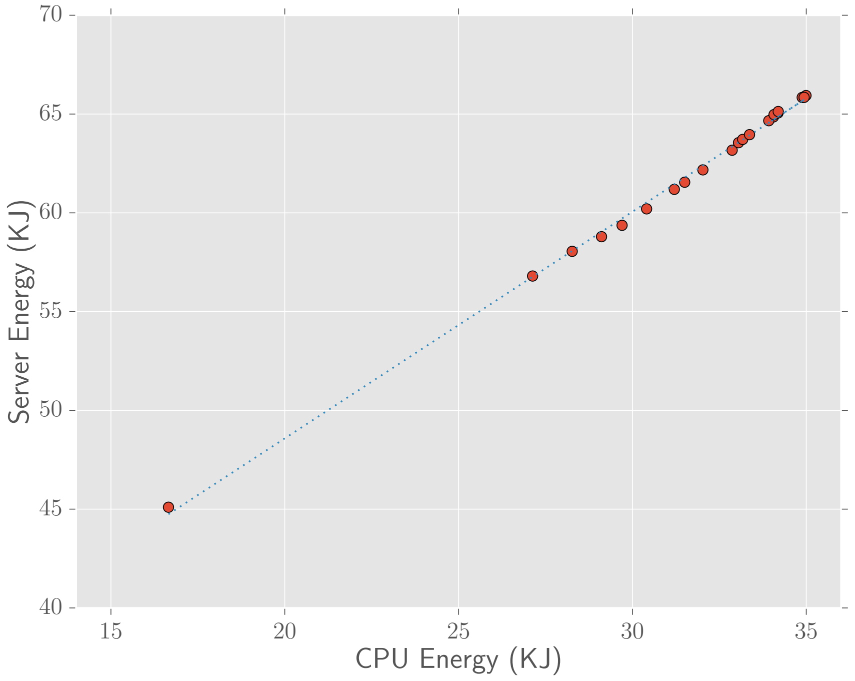


FIGURE C.1: Comparison of the CPU energy consumption measurements with RAPL versus the total server energy consumption measured with a power meter.

Bibliography

[1] Susanne Albers. "Online Scheduling". In: *Introduction to Scheduling* 3 (2009), pp. 51–73.

[2] Susanne Albers, Fabian Müller, and Swen Schmelzer. "Speed Scaling on Parallel Processors". In: *Proc. SPAA*. San Diego, USA: ACM, 2007, pp. 289–298.

[3] Alciom. *PowerSpy2: User Manual.* `https://goo.gl/a3nIwR`. Last visited: 2017-05-04. 2013.

[4] Jesse Alpert and Nissan Hajaj. *We knew the Web was big...* `https://googleblog.blogspot.it/2008/07/we-knew-web-was-big.html`. Last visited: 2016-11-04. 2008.

[5] Ioannis Arapakis, Xiao Bai, and B. Barla Cambazoglu. "Impact of Response Latency on User Behavior in Web Search". In: *Proc. SIGIR*. Gold Coast, Australia: ACM, 2014, pp. 103–112.

[6] Claudine Santos Badue, Ricardo Baeza-Yates, Berthier Ribeiro-Neto, Arthur Ziviani, and Nivio Ziviani. "Analyzing Imbalance among Homogeneous Index Servers in a Web Search System". In: *Information Processing & Management* 43.3 (2007). Special Issue on Heterogeneous and Distributed Information Retrieval, pp. 592–608.

[7] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology behind Search.* 2nd ed. Addison-Wesley, 2011.

[8] Ricardo Baeza-Yates, Aristides Gionis, Flavio Junqueira, Vassilis Plachouras, and Luca Telloli. "On the Feasibility of Multi-site Web Search Engines". In: *Proc. CIKM.* Hong Kong, China: ACM, 2009, pp. 425–434.

[9] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. "Speed Scaling to Manage Energy and Temperature". In: *Journal of the ACM* 54.1 (2007), pp. 1–39.

[10] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. "The Datacenter as a Computer: an Introduction to the Design of Warehouse-scale Machines". In: *Synthesis lectures on computer architecture* 8.3 (2013), pp. 1–154.

[11] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. "Web Search for a Planet: The Google Cluster Architecture". In: *IEEE Micro* 23.2 (2003), pp. 22–28.

[12] Luiz André Barroso and Urs Hölzle. "The Case for Energy-Proportional Computing". In: *Computer* 40.12 (2007), pp. 33–37.

[13] Roi Blanco, Matteo Catena, and Nicola Tonellotto. "Exploiting Green Energy to Reduce the Operational Costs of Multi-Center Web Search Engines". In: *Proc. WWW.* Montreal, Canada: IW3C2, 2016, pp. 1237–1247.

[14] Roi Blanco, B. Barla Cambazoglu, Flavio P. Junqueira, Ivan Kelly, and Vincent Leroy. "Assigning Documents to Master Sites in Distributed Search". In: *Proc. CIKM.* Glasgow, United Kingdom: ACM, 2011, pp. 67–76.

[15] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[16] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. "Efficient Query Evaluation Using a Two-level Retrieval Process". In: *Proc. CIKM.* New Orleans, USA: ACM, 2003, pp. 426–434.

[17] Dominik Brodowski. *CPU frequency and voltage scaling code in the Linux kernel.* `https://www.kernel.org/doc/Documentation/cpu-freq/index.txt`. Last visited: 2016-11-08. 2015.

[18] Stefan Büttcher and Charles L.A. Clarke. "Index Compression is Good, Especially for Random Access". In: *Proc. CIKM.* Lisbon, Portugal: ACM, 2007.

[19] Stefan Büttcher, Charles L.A. Clarke, and Gordon V. Cormack. *Information Retrieval: Implementing and Evaluating Search Engines.* MIT Press, 2010.

[20] B. Barla Cambazoglu and Ricardo Baeza-Yates. "Scalability Challenges in Web Search Engines". In: *Synthesis Lectures on Information Concept, Retrieval, and Services* 7.6 (2015), pp. 1–138.

[21] B. Barla Cambazoglu, Vassilis Plachouras, and Ricardo Baeza-Yates. "Quantifying Performance and Quality Gains in Distributed Web Search Engines". In: *Proc. SIGIR.* Boston, USA: ACM, 2009, pp. 411–418.

[22] B. Barla Cambazoglu, Emre Varol, Enver Kayaaslan, Cevdet Aykanat, and Ricardo Baeza-Yates. "Query Forwarding in Geographically Distributed Search Engines". In: *Proc. SIGIR.* Geneva, Switzerland: ACM, 2010, pp. 90–97.

[23] Matteo Catena, Craig Macdonald, and Iadh Ounis. "On Inverted Index Compression for Search Engine Efficiency". In: *Proc. ECIR.* Amsterdam, Netherlands: Springer, 2014, pp. 359–371.

[24] Matteo Catena, Craig Macdonald, and Nicola Tonellotto. "Load-sensitive CPU Power Management for Web Search Engines". In: *Proc. SIGIR.* Santiago, Chile: ACM, 2015, pp. 751–754.

[25] Matteo Catena and Nicola Tonellotto. "A Study on Query Energy Consumption in Web Search Engines." In: *Proc. IIR.* Cagliari, Italy: CEUR-WS, 2015, pp. 1–4.

[26] Matteo Catena and Nicola Tonellotto. "Energy-efficient Query Processing in Web Search Engines". In: *Transactions on Knowledge and Data Engineering* (2017).

[27] Sai Rahul Chalamalasetti, Martin Margala, Wim Vanderbauwhede, Mitch Wright, and Parthasarathy Ranganathan. "Evaluating FPGA-acceleration for real-time unstructured search". In: *Proc. ISPASS.* New Brunswick, USA: IEEE, 2012, pp. 200–209.

[28] Chronicle. *Thread Affinity: Minimise Jitter for Critical Threads.* `http://chronicle.software/products/thread-affinity/`. Last visited: 2017-05-04. 2016.

[29] Marco Danelutto, Daniele De Sensi, and Massimo Torquati. "Energy Driven Adaptivity in Stream Parallel Computations". In: *Proc. PDP.* Turku, Finland: IEEE, 2015, pp. 103–110.

[30] Daniele De Sensi. *Mammut: MAchine Micro Management UTilities.* `http://danieledesensi.github.io/mammut/`. Last visited: 2017-05-04. 2016.

[31] Daniele De Sensi. "Predicting Performance and Power Consumption of Parallel Applications". In: *Proc. PDP.* Heraklion, Greece: IEEE, 2016, pp. 200–207.

[32] Jeffrey Dean. “Challenges in Building Large-scale Information Retrieval Systems: Invited Talk”. In: *Proc. WSDM.* Barcelona, Spain: ACM, 2009, pp. 1–1.

[33] Jeffrey Dean and Luiz André Barroso. “The Tail at Scale”. In: *Communications of the ACM* 56.2 (2013), pp. 74–80.

[34] Constantinos Dimopoulos, Sergey Nepomnyachiy, and Torsten Suel. “Optimizing Top-k Document Retrieval Strategies for Block-max Indexes”. In: *Proc. WSDM.* Rome, Italy: ACM, 2013, pp. 113–122.

[35] Shuai Ding and Torsten Suel. “Faster Top-k Document Retrieval Using Block-max Indexes”. In: *Proc. SIGIR.* Beijing, China: ACM, 2011, pp. 993–1002.

[36] Shuai Ding, Jinru He, Hao Yan, and Torsten Suel. “Using Graphics Processors for High Performance IR Query Processing”. In: *Proc. WWW.* Madrid, Spain: ACM, 2009, pp. 421–430.

[37] Zhihui Du, Hongyang Sun, Yuxiong He, Yu He, David A. Bader, and Huazhe Zhang. “Energy-Efficient Scheduling for Best-Effort Interactive Services to Achieve High Response Quality”. In: *Proc. IPDPS.* Boston, USA: IEEE, 2013, pp. 637–648.

[38] Facebook. *Sustainability.* `https://sustainability.fb.com`. Last visited: 2017-03-17. 2017.

[39] Tiziano Fagni, Raffaele Perego, Fabrizio Silvestri, and Salvatore Orlando. “Boosting the Performance of Web Search Engines: Caching and Prefetching Query Results by Exploiting Historical Usage Data”. In: *Transactions on Information Systems* 24.1 (2006), pp. 51–78.

[40] Donald G. Fink and H. Wayne Beaty. *Standard Handbook for Electrical Engineers.* McGraw-Hill Engineering & Technology Management. McGraw-Hill, 1999.

[41] Marcus Fontoura, Vanja Josifovski, Jinhui Liu, Srihari Venkatesan, Xiangfei Zhu, and Jason Y. Zien. “Evaluation Strategies for Top-k Queries over Memory-Resident Inverted Indexes”. In: *PVLDB* 4.12 (2011), pp. 1213–1224.

[42] Ana Freire, Craig Macdonald, Nicola Tonellotto, Iadh Ounis, and Fidel Cacheda. “A Self-adapting Latency/Power Tradeoff Model for Replicated Search Engines”. In: *Proc. WSDM.* New York, USA: ACM, 2014, pp. 13–22.

[43] Ana Freire, Craig Macdonald, Nicola Tonellotto, Iadh Ounis, and Fidel Cacheda. “Hybrid Query Scheduling for a Replicated Search Engine”. In: *Proc. ECIR.* Moscow, Russia: Springer, 2013, pp. 435–446.

[44] Qingqing Gan and Torsten Suel. “Improved Techniques for Result Caching in Web Search Engines”. In: *Proc. WWW.* Madrid, Spain: ACM, 2009, pp. 431–440.

[45] Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, Jeffrey O. Kephart, and Charles Lefurgy. “Power Capping via Forced Idleness”. In: *Proc. WEED at ISCA.* Austin, USA: ACM, 2009, pp. 1–7.

[46] James Glanz. “Power, pollution and the internet”. In: *The New York Times* 22 (2012).

[47] Íñigo Goiri, Kien Le, Md. E. Haque, Ryan Beauchea, Thu D. Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. “GreenSlot: Scheduling Energy Consumption in Green Datacenters”. In: *Proc. SC.* Seattle, USA: ACM, 2011, 20:1–20:11.

[48] Ben Gomes. *Google Instant, behind the scenes.* `https://googleblog.blogspot.it/2010/09/google-instant-behind-scenes.html`. Last visited: 2017-03-16. 2010.

[49] Google. *Efficiency: How we do it.* `https://www.google.com/about/datacenters/efficiency/internal/`. Last visited: 2017-03-20. 2016.

[50] Google. *Environment.* `https://environment.google/`. Last visited: 2017-03-17. 2017.

[51] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. "The Cost of a Cloud: Research Problems in Data Center Networks". In: *SIGCOMM Computer Commununication Review* 39.1 (2008), pp. 68–73.

[52] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. "An Energy Efficiency Feature Survey of the Intel Haswell Processor". In: *Proc. IPDPSW.* Hyderabad, India: IEEE, 2015, pp. 896–904.

[53] Mor Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action.* Cambridge University Press, 2013.

[54] Vassiliki Hatzi, B. Barla Cambazoglu, and Iordanis Koutsopoulos. "Optimal Web Page Download Scheduling Policies for Green Web Crawling". In: *IEEE Journal on Selected Areas in Communications* 34.5 (2016), pp. 1378–1388.

[55] Vassiliki Hatzi, B. Barla Cambazoglu, and Iordanis Koutsopoulos. "Web Page Download Scheduling Policies for Green Web Crawling". In: *Proc. SoftCOM.* Split, Croatia: IEEE, 2014, pp. 56–60.

[56] Nicolas Hidalgo, Erika Rosas, Veronica Gil-Costa, and Mauricio Marin. "Assessing Energy Efficiency in ISP and Web Search Engine Collaboration". In: *Proc. WAINA.* Victoria, Canada: IEEE, 2014, pp. 299–304.

[57] Bradley Huffaker, Marina Fomenkov, Daniel J. Plummer, David Moore, and Kimberly Claffy. "Distance Metrics in the Internet". In: *Proc. ITS.* Natal, Brazil: IEEE, 2002, pp. 200–202.

[58] Vijay Janapa Reddi, Benjamin C. Lee, Trishul Chilimbi, and Kushagra Vaid. "Web Search Using Mobile Cores: Quantifying and Mitigating the Price of Efficiency". In: *Proc. ISCA.* Saint-Malo, France: ACM, 2010, pp. 314–325.

[59] Myeongjae Jeon, Saehoon Kim, Seung-won Hwang, Yuxiong He, Sameh Elnikety, Alan L. Cox, and Scott Rixner. "Predictive Parallelization: Taming Tail Latencies in Web Search". In: *Proc. SIGIR.* Gold Coast, Australia: ACM, 2014, pp. 253–262.

[60] Flavio P. Junqueira, Vincent Leroy, and Matthieu Morel. "Reactive Index Replication for Distributed Search Engines". In: *Proc. SIGIR.* Portland, USA: ACM, 2012, pp. 831–840.

[61] Enver Kayaaslan, B. Barla Cambazoglu, Roi Blanco, Flavio P. Junqueira, and Cevdet Aykanat. "Energy-price-driven Query Processing in Multi-center Web Search Engines". In: *Proc. SIGIR.* Beijing, China: ACM, 2011, pp. 983–992.

[62] Saehoon Kim, Yuxiong He, Seung-won Hwang, Sameh Elnikety, and Seungjin Choi. "Delayed-Dynamic-Selective (DDS) Prediction for Reducing Extreme Tail Latency in Web Search". In: *Proc. WSDM.* Shanghai, China: ACM, 2015, pp. 7–16.

[63] Jonathan G. Koomey, Christian Belady, Christian Patterson, and Christian Santos. *Assessing Trends over Time in Performance, Costs, and Energy Use for Servers.* `http://www.intel.com/assets/pdf/general/servertrendsreleasecomplete-v25.pdf`. Last visited: 2016-11-04. 2009.

[64] Kien Le, Ricardo Bianchini, Thu D. Nguyen, Ozlem Bilgir, and Margaret Martonosi. "Capping the Brown Energy Consumption of Internet Services at Low Cost". In: *Proc. GREENCOMP.* Chicago, USA: IEEE, 2010, pp. 3–14.

[65] Daniel Lemire and Leonid Boytsov. "Decoding Billions of Integers per Second Through Vectorization". In: *Software: Practice and Experience* 45.1 (2015), pp. 1–29.

[66] Ronny Lempel and Fabrizio Silvestri. "Web Search Result Caching and Prefetching". In: *Encyclopedia of Database Systems.* Springer, 2009, pp. 3501–3506.

[67] Lemur Project. *The ClueWeb09 Dataset.* `http://lemurproject.org/clueweb09`. Last visited: 2016-11-08. 2009.

[68] Jacob Leverich and Christos Kozyrakis. "On the Energy (in)Efficiency of Hadoop Clusters". In: *SIGOPS Operating Systems Review* 44.1 (2010), pp. 61–65. ISSN: 0163-5980.

[69] Kenan Liu, Gustavo Pinto, and Yu David Liu. "Data-Oriented Characterization of Application-Level Energy Optimization". In: *Proc. FASE at ETAPS.* London, United Kingdom: Springer, 2015, pp. 316–331.

[70] Tie-Yan Liu. "Learning to Rank for Information Retrieval". In: *Foundations and Trends in Information Retrieval* 3.3 (2009), pp. 225–331.

[71] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. "Towards Energy Proportionality for Large-scale Latency-critical Workloads". In: *Proc. ISCA.* Minneapolis, USA: ACM, 2014, pp. 301–312.

[72] Yung-Cheng Ma, Tien-Fu Chen, and Chung-Ping Chung. "Posting File Partitioning and Parallel Information Retrieval". In: *Journal of Systems and Software* 63.2 (2002), pp. 113–127.

[73] Yung-Cheng Ma, Chung-Ping Chung, and Tien-Fu Chen. "Load and Storage Balanced Posting File Partitioning for Parallel Information Retrieval". In: *Journal of Systems and Software* 84.5 (2011), pp. 864–884.

[74] Craig Macdonald, Rodrygo L.T. Santos, and Iadh Ounis. "The Whens and Hows of Learning to Rank for Web Search". In: *Information Retrieval* 16.5 (2013), pp. 584–628.

[75] Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. "Learning to Predict Response Times for Online Query Scheduling". In: *Proc. SIGIR.* Portland, USA: ACM, 2012, pp. 621–630.

[76] Craig Macdonald, Richard McCreadie, Rodrygo L.T. Santos, and Iadh Ounis. "From Puppy to Maturity: Experiences in Developing Terrier". In: *Proc. OSIR at SIGIR* (2012), pp. 60–63.

[77] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[78] Mauricio Marin, Veronica Gil-Costa, and Carlos Gomez-Pantoja. "New Caching Techniques for Web Search Engines". In: *Proc. HPDC.* Chicago, USA: ACM, 2010, pp. 215–226.

[79] David Meisner, Brian T. Gold, and Thomas F. Wenisch. “PowerNap: Eliminating Server Idle Power”. In: *Proc. ASPLOS*. Washington, USA: ACM, 2009, pp. 205–216.

[80] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. “Power Management of Online Data-intensive Services”. In: *Proc. ISCA*. San Jose, USA: ACM, 2011, pp. 319–330.

[81] Microsoft. *Environmental Sustainability at Microsoft.* `https://www.microsoft.com/about/csr/environment/`. Last visited: 2017-03-17. 2017.

[82] Microsoft. *Microsoft 2015 Citizenship Report.* `http://bit.ly/1TCFkGP`. Last visited: 2017-03-20. 2015.

[83] Microsoft. *MSN 2006 Query Log.* `http://research.microsoft.com/en-us/um/people/nickcr/wscd09`. Last visited: 2016-11-08. 2009.

[84] Sparsh Mittal. “Power Management Techniques for Data Centers: A Survey”. In: *CoRR* abs/1404.6681 (2014).

[85] Alistair Moffat, William Webber, Justin Zobel, and Ricardo Baeza-Yates. “A Pipelined Architecture for Distributed Text Query Evaluation”. In: *Information Retrieval* 10.3 (2007), pp. 205–231.

[86] Anne-Cecile Orgerie, Marcos Dias de Assuncao, and Laurent Lefevre. “A Survey on Techniques for Improving the Energy Efficiency of Large-scale Distributed Systems”. In: *ACM Computing Surveys* 46.4 (2014), pp. 1–31.

[87] Giuseppe Ottaviano, Nicola Tonellotto, and Rossano Venturini. “Optimal Space-time Tradeoffs for Inverted Indexes”. In: *Proc. WSDM*. Shanghai, China, 2015, pp. 47–56.

[88] Venkatesh Pallipadi, Shaohua Li, and Adam Belay. “cpuidle: Do nothing, efficiently”. In: *Proc. Linux Symposium*. Ottawa, Canada, 2007, pp. 119–125.

[89] Nord Pool. *Market Data.* `http://www.nordpoolspot.com`. Last visited: 2015-06-01. 2015.

[90] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. “A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services”. In: *Proc. ISCA*. Minneapolis, USA: ACM, 2014, pp. 13–24.

[91] Asfandyar Qureshi, Rick Weber, Hari Balakrishnan, John Guttag, and Bruce Maggs. “Cutting the Electric Bill for Internet-scale Systems”. In: *Proc. SIGCOMM*. Barcelona, Spain: ACM, 2009, pp. 123–134.

[92] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. “No "Power" Struggles: Coordinated Multi-level Power Management for the Data Center”. In: *Proc. ASPLOS*. Seattle, USA: ACM, 2008, pp. 48–59.

[93] Parthasarathy Ranganathan, Phil Leech, David Irwin, and Jeffrey Chase. “Ensemble-level Power Management for Dense Blade Servers”. In: *Proc. ISCA*. Boston, USA: ACM, 2006, pp. 66–77.

[94] Shaolei Ren, Yuxiong He, Sameh Elnikety, and Kathryn S. McKinley. “Exploiting Processor Heterogeneity in Interactive Services”. In: *Proc. ICAC*. San Jose, USA: USENIX, 2013, pp. 45–58.

[95] Stephen Robertson and Hugo Zaragoza. “The Probabilistic Relevance Framework: BM25 and Beyond”. In: *Foundations and Trends in Information Retrieval* 3.4 (2009), pp. 333–389.

[96] Fethi Burak Sazoglu, B. Barla Cambazoglu, Rifat Ozcan, Ismail Sengor Altingovde, and Özgür Ulusoy. “A Financial Cost Metric for Result Caching”. In: *Proc. SIGIR*. Dublin, Ireland: ACM, 2013, pp. 873–876.

[97] Eric Schurman and Jake Brutlag. “Performance Related Changes and their User Impact”. In: *Proc. Velocity*. San Jose, USA: O’Reilly, 2009.

[98] Fabrizio Silvestri. “Mining Query Logs: Turning Search Usage Data into Knowledge”. In: *Foundations and Trends in Information Retrieval* 4.1–2 (2010), pp. 1–174.

[99] David C. Snowdon, Sergio Ruocco, and Gernot Heiser. “Power Management and Dynamic Voltage Scaling: Myths and Facts”. In: *Proc. PARC workshop at EMSoft*. IEEE, 2005.

[100] Christopher Stewart and Kai Shen. “Some Joules are More Precious Than Others: Managing Renewable Energy in the Datacenter”. In: *Proc. HotPower workshop at SOSP*. Big Sky, USA: ACM, 2009.

[101] Amin Teymorian, Ophir Frieder, and Marcus A. Maloof. “Rank-energy Selective Query Forwarding for Distributed Search Systems”. In: *Proc. CIKM*. San Francisco, USA: ACM, 2013, pp. 389–398.

[102] The Climate Group for the Global e-Sustainability Initiative. *Smart 2020: Enabling the low carbon economy in the information age*. `http://gesi.org/files/Reports/Smart%202020%20report%20in%20English.pdf`. Last visited: 2016-11-04. 2008.

[103] The Linux Kernel Archives. *Intel P-State driver*. `https://www.kernel.org/doc/Documentation/cpu-freq/intel-pstate.txt`. Last visited: 2016-11-08. 2016.

[104] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. “Efficient and Effective Retrieval Using Selective Pruning”. In: *Proc. WSDM*. Rome, Italy: ACM, 2013, pp. 63–72.

[105] Howard Turtle and James Flood. “Query Evaluation: Strategies and Optimizations”. In: *Information Processing & Management* 31.6 (1995), pp. 831–850.

[106] U.S. Department of Energy. *Quick Start Guide to Increase Data Center Energy Efficiency*. `http://www.gsa.gov/graphics/pbs/data_center_quick_start_03_09_508_compliant.pdf`. Last visited: 2016-11-04. 2009.

[107] Sebastiano Vigna. “Quasi-succinct Indices”. In: *Proc. WSDM*. Rome, Italy: ACM, 2013, pp. 83–92.

[108] Amos Waterland. *stress: a Tool to Impose Load on and Stress Test Systems*. `http://people.seas.harvard.edu/~apw/stress/`. Last visited: 2017-05-04. 2014.

[109] Wikipedia. *Electricity pricing*. `https://en.wikipedia.org/wiki/Electricity_pricing`. Last visited: 2015-06-01. 2015.

[110] Ian H. Witten, Timothy C. Bell, and Alistair Moffat. *Managing Gigabytes: Compressing and Indexing Documents and Images*. 1st. New York, NY, USA: Wiley, 1994.

[111] WorldWideWebSize.com. *The size of the World Wide Web (The Internet).* `http://www.worldwidewebsize.com`. Last visited: 2016-11-04. 2016.

[112] Hao Wu and Hui Fang. “Analytical Performance Modeling for Top-K Query Processing”. In: *Proc. CIKM.* Shanghai, China: ACM, 2014, pp. 1619–1628.

[113] Jing Yan, Zhan-Xiang Zhao, Ning-Yi Xu, Xi Jin, Lin-Tao Zhang, and Feng-Hsiung Hsu. “Efficient Query Processing for Web Search Engine with FPGAs”. In: *Proc. FCCM.* Toronto, Canada: IEEE, 2012, pp. 97–100.

[114] Frances Yao, Alan Demers, and Scott Shenker. “A Scheduling Model for Reduced CPU Energy”. In: *Proc. FOCS.* Washington, USA: IEEE, 1995, pp. 374–382.