UNIVERSIDADE DA CORUÑA

DEPARTMENT OF INFORMATION

AND COMMUNICATIONS TECHNOLOGIES

Query Scheduling Techniques and Power/Latency Trade-off Model for Large-Scale Search Engines

PHD THESIS

Ana María Freire Veiga

PhD Supervisor: Dr. Fidel Cacheda Seijo

2014

UNIVERSIDADE DA CORUÑA

PhD Thesis

Query Scheduling Techniques and Power/Latency Trade-off Model for Large-Scale Search Engines

Ana María Freire Veiga

PhD supervisor:

Dr. Fidel Cacheda Seijo

Thesis committee:

Dr. Martín Llamas Nistal

Dr. Raffaele Perego

Dr. Sergio Guilherme Aleixo de Matos

Dr. Juan Manuel Fernández Luna

Dr. Manuel Álvarez Díaz

D. Fidel Cacheda Seijo, Doctor by the University of A Coruña

D. Fidel Cacheda Seijo, Doctor por la Universidad de A Coruña

CERTIFIES

CERTIFICA

That this thesis report titled ***Query Scheduling Techniques and Power/Latency Trade-off Model for Large-Scale Search Engines*** was developed by **Dª. Ana María Freire Veiga** under his direction in order to obtain the International Doctor mention by the University of A Coruña.

Que la presente memoria titulada **Query Scheduling Techniques and Power/Latency Trade-off Model for Large-Scale Search Engines** *ha sido realizada bajo su dirección y constituye la Tesis que presenta* ***Dª. Ana María Freire Veiga*** *para optar al grado de Doctor con mención Internacional por la Universidad de A Coruña.*

A Coruña, 2014

Ana María Freire Veiga	Dr. Fidel Cacheda Seijo
(PhD Student)	(Thesis Supervisor)
(*Doctoranda*)	(*Director de la Tesis*)

A mis padres, mi constante apoyo.

Acknowledgments

This path would not have been possible without the help of my supervisor, Fidel Cacheda, who gave me the opportunity of diving again into the Information Retrieval field, after an unforgettable start in the Information Retrieval Lab. Thank you for being such a great motivating person, for guiding and advising me in the best way and, above all, for always letting me choose the next step.

All my gratitude to all the colleagues that have shared with me great moments in the Telematic Engineering Lab. Thank you for making our office a warm place to spend lots of hours and for building such a great friendship even out of our blind walls.

I would like to acknowledge Iadh Ounis and the Information Retrieval Group from the University of Glasgow. They gave me the opportunity of staying three months in 2012 in one of the leading groups in IR, where I put in touch with an impeccable way of work. My special gratitude goes to Craig Macdonald: thank you for becoming my best teacher and reference during all my PhD. I can't ever forget some of your encouraging words that became my motto: *Don't think "is this sufficient?" but "how can we do better"*. All this work would not have being possible without your valuable help.

Vorrei ringraziare in modo particolare a tutti i membri del High Performance Computing Lab di Pisa. Per me è stato un vero piacere condividere con voi quattro mesi, me avete fatto sentire ogni giorno come se fossi a casa mia. Grazie Nicola Tonellotto per invitarmi a lavorare con te e per avermi permesso di capire quanto importante è avere un capo esigente.

A special gratitude also for Silvia Lorenzo Freire, for sharing with me her huge knowledge and offering me her useful help.

I can not forget Roi Blanco, who was discreetly present at every milestone of my career. Thank you also for opening me the next opportunity of learning, I will make the most of it.

A great deal of gratitude is due to all the people that have walked with me during this period. Your encouraging and warm-hearted words were the best incentive to finish this work. Those who kept me out of this thesis, sharing awesome moments and messages, have also done a good job.

Mis mayores palabras de agradecimiento son para mi familia, por haber estado siempre a mi lado, incluso cuando mis obligaciones ponían grandes distancias de por medio. Vosotros habéis construido la base de todo esto, yo sólo he intentado aprovecharla lo máximo posible. Os estaré agradecida siempre.

Abstract

Web search engines have to deal with a rapid increase of information, demanded by high incoming query traffic. This situation has driven companies to build geographically distributed data centres housing thousands of computers, consuming enormous amounts of electricity and requiring a huge infrastructure around. At this scale, even minor efficiency improvements result in large financial savings.

This thesis represents a novel contribution to query scheduling and power consumption state-of-the-art, by assisting large-scale data centres to build more efficient search engines.

On the one hand, this thesis proposes new scheduling techniques to decrease the response time of queries, by estimating the server that will be idle soonest.

On the other hand, this thesis defines a simple mathematical model that establishes a threshold between the power and latency of a search engine. Using historical and current data, the model estimates the incoming query traffic and automatically increases/decreases the necessary number of active machines in the system. We achieve high energy savings during the whole day, without degrading the latency.

Our experiments have attested the power of both scheduling methods and the power/latency trade-off model in improving the efficiency and achieving high energy savings.

Resumen

Los motores de búsqueda actuales deben enfrentarse a un veloz incremento de información y a un enorme tráfico de consultas. Las grandes compañías se han visto obligadas a construir centros de datos geográficamente distribuidos y compuestos por miles de servidores. El suministro eléctrico supone un enorme gasto energético, por lo que una pequeña mejora a nivel de eficiencia puede suponer grandes ventajas económicas.

Esta tesis permitirá a grandes compañías de Recuperación de Información la construcción de motores de búsqueda dotados de mayor eficiencia.

Por una parte, esta tesis propone nuevas técnicas de distribución de consultas a los servidores que las procesan para disminuir su tiempo de respuesta, estimando cuál será el primer servidor disponible.

Por otra parte, esta tesis define un modelo matemático que establece un balance entre el tiempo de respuesta de un motor de búsqueda y su consumo energético. Basándonos en datos históricos y actuales, el modelo estima el tráfico de consultas entrante y, de modo automático, aumenta/disminuye los servidores necesarios para procesar las consultas. Se consigue así un gran porcentaje de ahorro energético sin degradar la latencia del sistema.

Nuestros experimentos atestiguan las grandes mejoras alcanzadas en cuanto a eficiencia y ahorro energético.

Resumo

Os motores de busca actuais deben enfrontarse a un grande incremento de información e a un enorme tráfico de consultas. As grandes compañías víronse obrigadas a construír centros de datos xeograficamente distribuídos e compostos por milleiros de servidores. A subministración eléctrica supón un enorme gasto enerxético, polo que una pequena mellora a nivel de eficiencia pode supoñer grandes vantaxes económicas.

Esta tese permitirá a grandes compañías de Recuperación de Información a construción de motores de busca dotados de maior eficiencia.

Por una parte, esta tese propón novas técnicas de distribución de consultas aos servidores que as procesan para diminuír su tempo de resposta, estimando cál será o primeiro servidor dispoñible.

Por outra parte, esta tese define un modelo matemático que establece un balance entre o tempo de resposta dun motor de busca e o seu consumo enerxético. A partir de datos históricos e actuais, o modelo estima o tráfico de consultas entrantes e automaticamente aumenta/diminúe os servidores necesarios para procesar as consultas. Conséguese así unha grande porcentaxe de aforro enerxético sen degradar a latencia do sistema.

Os nosos experimentos testemuñan as grandes melloras alcanzadas en canto a eficiencia e aforro enerxético.

Index

List of Tables

List of Figures

Chapter 1

Thesis Outline

1.1 Introduction

The rapid evolution of Internet has turned the World Wide Web into the hugest information repository ever: the indexed Web by Google contains nearly 20 billion pages[1] and the users rise 2.4 billions of people (34% of the world population)[2].

This great amount of information is managed by Web Search Engines, that have evolved their infrastructures into large-scale data centers in order to solve the queries rapidly. This ambitious objective usually confront two important factors in Information Retrieval (IR) systems: latency and power consumption. Typically, when the queries need to be answered faster, more machines are added to the system in order to decrease waiting time before entering the query servers. On the other hand, decreasing power consumption comes with an increasing of query response time.

The main objective of this thesis is, on the one hand, proposing novel scheduling techniques that improve the state-of-the-art approaches and reduce the response time of the queries and, on the other hand, defining a new mathematical model that automatically establishes a trade-off between latency and power consumption of a large-scale data center. This way, the novel mathematical model aims to save energy consumption (with the corresponding financial savings) without compromising the efficiency of the system. The duel between efficiency and energy consumption seems to become an agreement.

[1] `http://worldwidewebsize.com` - Last update: 2014.
[2] `http://internetworldstats.com` - Last update: 2012.

1.2 Motivation

Our research is mainly motivated by the following facts:

- There is not so much query scheduling related state-of-the art, and many of the published techniques assume some inadequate conditions that give rise to an inefficient scheduling.
- Some IR companies are strongly taking care about their energy consumption. They are increasing the use of renewable energy and more efficient systems in order to save energy, reduce carbon and cut IT costs, following *Green IR* advices.
- It would be really interesting to expand this green behavior to more IR companies to address environmental challenges and achieve sustainability.
- To the best of our knowledge, there is no previous work in the state-of-the-art in charge of managing the number of active query servers into a search engine.
- Reducing the waiting time of queries as well as the power consumption also has an effect on the financial costs of the companies, and this constitutes a great incentive.

1.3 Thesis Statement

The statements of this thesis are the following:

- The efficiency of distributed and replicated Information Retrieval system can be improved by means of query efficiency predictors for performing the scheduling.
- Power consumption of large-scale search engines can be reduced without compromising their efficiency.

These issues are addressed separately on this thesis, but both of them are oriented to the same objective: improve the efficiency of the whole system. The first issue focuses on selecting the most appropriate replica to serve each query, in order to improve the efficiency of the system. The second issue aims to reduce the number of replicas used by a data center to answer the incoming queries, based on query traffic estimations, without degrading the latency of the search engine.

1.4 Contributions

The main contributions of this work are the following:

- We attest that simulation platforms are reliable for IR experimentation, leading to resource savings. We support this conclusion by establishing a complete analysis of the current IR evaluation platforms.
- We introduce query efficiency predictors as suitable estimators to improve query scheduling. We develop a new scheduling method for choosing the replica with the fastest queue to process a query. This method estimates the processing time of the queued queries and it calculates an approximate time that a new query must spend in the queue of each replica. Based on this estimation, our method selects the more suitable replica.
- We also combine the previous method with more simpler scheduling techniques, developing a new hybrid scheduling method that avoids the overhead inherent to query predictors calculation and improves the state-of-the art.
- Once we have developed new methods to improve the response time of a search engine, we focus on reducing the power consumption of the whole system. We propose a mathematical model that establishes a trade-off between latency and power consumption. This model attempts to automatically adapt the number of active replicas in the system based on incoming query traffic. Results attest that our model allows to achieve high energy savings without compromising the efficiency of the system.
- We prove the limitation of $M/M/s$ Queueing Theory model for estimating the latency in search engines. As a consequence, we develop the previous model by predicting the latency using historical data, attesting the good performance of this approach.

1.5 Origins of the Material

The material that forms parts of this thesis has found their origins in various conference papers that have been published during the course of the PhD research. In particular:

- Contents in Chapter 3 about the comparison among different IR evaluation platforms have been published as a contribution to a workshop: *Analysis of performance evaluation techniques for Large Scale Information Retrieval*[41]. Ana Freire, Fidel Cacheda, Vreixo Formoso and Víctor Carneiro. In Proceedings of LSDS-IR 2013.

- The query scheduling method based on predictors and presented in Chapter 4 has been published in one of the leading IR conferences (A*): *Scheduling Queries Across Replicas* [42]. Ana Freire, Craig Macdonald, Nicola Tonellotto, Iadh Ounis and Fidel Cacheda. In Proceedings of SIGIR 2012. (36.5% acceptance).

- The Hybrid query scheduling technique writen in Chapter 5 have been published as a full paper: *Hybrid query scheduling for a replicated search engine* [43]. Ana Freire, Craig Macdonald, Nicola Tonellotto, Iadh Ounis and Fidel Cacheda. In Proceedings of ECIR 2013. (29% acceptance).

- The last part of the thesis, that constitutes the Chapter 6 has been recently accepted as the following full paper: *A Self-Adapting Latency/Power Tradeoff Model for Replicated Search Engines*. Ana Freire, Craig Macdonald, Nicola Tonellotto, Iadh Ounis and Fidel Cacheda. In Proceedings of WSDM 2014. (18% acceptance).

1.6 Thesis Outline

The novel contributions of this thesis are presented in chapters 3, 4, 5 and 6. Chapter 2 introduces basic concepts of Information Retrieval for a non Information Retrieval expert. The organization of the following chapters is as follows:

- Chapter 2 introduces the concepts from IR that this thesis relies on. In particular, concepts from the general IR process as indexing and retrieval are introduced. We describe how IR systems were evolved into the advent of the Web and how the distributed systems become necessary to manage all the available information. This chapter ends by introducing the task of IR evaluation and presents the IR test collections used by the community to evaluate and easily compare the implemented systems.

- Chapter 3 examines the state-of-the-art IR evaluation platforms and establishes an exhaustive comparison among them, considering financial costs, difficulty in developing the evaluation platform and reliability of the results. We develop and detail some experimentation in order to compare the suitability of the evaluated platforms for representing a real search engine.

- Chapter 4 investigates in detail how query efficiency predictors can be used to propose a new query scheduling method, namely *Least Loaded*, that improves the current state-of-the art techniques. We firstly introduce the tasks of query scheduling and dynamic pruning to later explain the function of query efficiency predictors, that constitute the key concept of the first part of this thesis. Experimentation is carried out using two datasets in order to test the behavior of the proposed method under different number of queries, variable incoming query flow, synthetic and also real query logs. An exhaustive study of the results and some critical conclusions form the end of this chapter and give way to the following one.

- Chapter 5 deals with the drawbacks of the Least Loaded method and propose a new *Hybrid* scheduling method that combines Least Loaded with previous existing techniques, as Round Robin, with the aim of exploiting the main advantages of each methods. In particular, we develop a hybrid scheduling method that performs as Least Loaded when the incoming query traffic is high, and imitates a lighter method as Round Robin when the contention is low. This way, the method takes advantage of the efficient features of query time prediction, but uses a simpler method that does not include any delay for calculating predictors.

- Chapter 6 is motivated by the concept of *Green IR*, that refers to the environment sustainability in large IR data centers. We define a power/latency trade-off mathematical model whose aim is to save power consumption when there is low contention in the system and not all the query servers need to be activated. Therefore, the system automatically turns query servers on or standby depending on the incoming query traffic. Experiments with a real daily query log allow to conclude the percentages of energy savings into a data centre.

- Chapter 7 closes this thesis with the conclusions drawn from this work, as well as possible directions of future work across the investigated tasks.

Chapter 2

Introduction

2.1 Introduction

This chapter introduces the reader to the field of Information Retrieval (IR). Section 2.2 starts with a brief explanation about the Information Retrieval process, including indexing and searching tasks. The evolution of the World Wide Web incurs the application of IR techniques to retrieve information from on-line repositories, and this fact cause the birth of the term Web Information Retrieval, described in Section 2.3. The rapid growth of the Web forces to deploy the IR systems upon large infrastructures, described in Section 2.4. Such distributed architectures entail high energy consumption that leads to financial cost raising. Green IR is an arising field that is in charge of promoting good habits into IR datacentres. This term and some state-of-the art works that tackle with power consumption are presented in Section 2.5. Finally, section 2.6 introduces the task of IR evaluation, and gives way to Chapter 3.

2.2 Information Retrieval

The confusion between data and information retrieval is a general perception. In data retrieval we usually want to check if an item is or is not present in the file. In information retrieval we normally want to find those items which partially match a user request and then select the best matching ones [84].

Information Retrieval deals with the representation, storage, organization of and

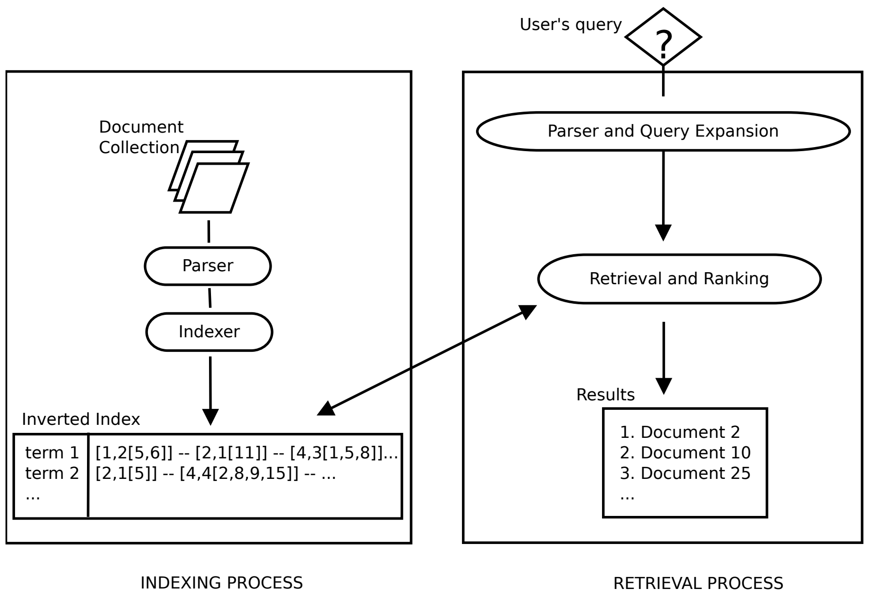


Figure 2.1: General scheme of an Information Retrieval System.

access to information items [9]. The whole process aims to solve a user *information need*, that she/he manifests in the form of a *query* (bag of keywords). The IR system is in charge of retrieving the items that are *relevant* to the user's information need. The user satisfaction is measured in terms of the relevance of the retrieved items and the time that the system spends in retrieving them. This way, if some non-relevant items are *ranked* higher than the relevant ones, the answer is unsatisfactory [64].

Accordingly, we can state the objective of an IR system as follows: to retrieve relevant items to satisfy the user's information need and rank these items higher than non-relevant ones.

These pieces of information that we refer to as items are usually in the form of documents [84], but these documents can represent web pages, book chapters, emails, and even non-textual information such as multimedia data (images, videos and music).

The general process of Information Retrieval is represented in figure 2.1.

The first task is processing the documents that will be indexed, i.e., *stopwords* (common terms with low semantic content, such as prepositions) are filtered out and some verbs are reduced to their stem (*stemming*). The resulting indexing terms will be used to build an index of the collection. An index is a critical data structure because it allows fast searching over large volumes of data. The most popular index structure is an inverted index [10]. Its basic structure is composed by the *vocabulary* and the *occurrences*. The vocabulary corresponds to the set of words previously obtained after the processing of the documents. The occurrences correspond to the documents that contain each word of the vocabulary. If we refer to a full inverted index, the structure also includes the positions of the tems of the vocabulary into each document (see Figure 2.1).

Once the indices are built (indexing process), the retrieval process can be started. When a query arrives at the system, it is parsed in the same way the documents were (stopwords, stemming...). Additionally, the query expansion process is applied to the incoming query. This process reformulates the query using synonyms or correcting errors to improve the retrieval process.

Subsequently, the retrieval process is performed, by searching the terms of the reformulated query across the indices. Different term-weighting schemes derived from different models of retrieval can be followed to compute a *score* of each document (i.e. *BM-25* [85], derived from the probabilistic model).

The system sorts the documents based on the estimated importance to the user (score) and straightaway it presents the *ranking* of final documents.

Two key terms are defined in the retrieval process: *efficiency* and *effectiveness*. Efficiency is mostly concerned with resources used to answer the query, and it is usually related to the time spent in giving an answer to the user. Effectiveness refers to the quality of the retrieved documents regarding their appropriateness to the user's information need. This thesis is concerned with the term efficiency, in the sense that we try to reduce the response time of the search engines at the same time we propose to save resources, specially in the field fo Web Information Retrieval.

2.3 Web Information Retrieval

The World Wide Web (Web) was created by Berners-Lee [12] in the early 90's at CERN[1] for sharing research documents. From this modest purpose, the Web has become the largest repository ever, making our life unthinkable without it.

[1]CERN: European Organization for Nuclear Research (`http://home.web.cern.ch/`)

The incredible leap of the Web has created an explosion in the field of Information Retrieval. The diversity of the web and the fact that everyone can generate content leads to the necessity of efficiently manage the on-line information, but the fast-changing of this kind of material makes it very different from the traditional collections. Web search engines have been (and continue) adapting their platforms to this such amount of information. Yahoo[2], Google[3] or Yandex[4] are examples of wold famous search engines (the third one is stomping in Russia) that answer millions of queries daily - Google was reported to answer more than one billion queries per day [5].

The nature of the Web incurs several challenges for IR systems [10, 25]:

- Distributed data: web pages are stored in geographically distributed servers, what makes the access of information an arduous task.
- Huge data volume: thousands of documents are stored and increase everyday.
- Constant change of data: electronic documents are subject to changes with more or less frequency and they can also be deleted.
- Unstructured data: web pages do not follow an unique structure.
- Quality of data: in most of the cases there is not an editorial process, and this lack of control causes lots of errors: false information, typing mistakes, etc.
- Heterogeneous data: different kind of files (i.e. videos, photos or text), different languages and encodings make the web an heterogeneous data source.

In order to deal with many of these obstacles, a crucial module is included at the beginning of the IR process: *crawler* (see Figure 2.2). A *Web Crawler*, *Web Spider* or *Web Robot* is a software for downloading pages from the Web automatically [10]. Web Crawlers collect web pages by sending requests to Web Servers across the network. Later, these web pages will be used as the collection to index and search by an IR system.

If we add the crawling module to Figure 2.1, we obtain the general scheme for Web IR (Figure 2.2).

[2] `http://www.yahoo.com`
[3] `http://www.google.com`
[4] `http://www.yandex.com`
[5] `http://www.google.com/competition/howgooglesearchworks.html`.

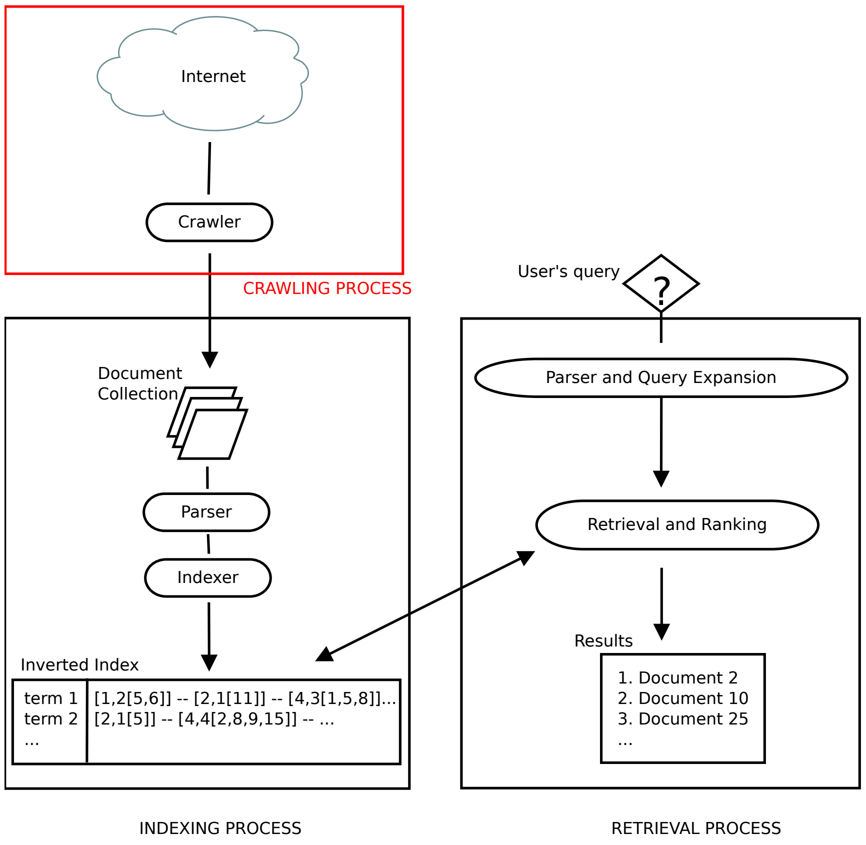


Figure 2.2: General scheme of a Web Information Retrieval System.

Crawlers need to tackle with several issues such as hidden pages (they can not be reached through links), duplicates (various URLs referring to the same content) or Soft-404 pages (custom error pages), that make the crawling process a non-trivial task, that is why efficient crawling has become essential to cope with the fast change of the Web content.

Apart from the crawling process, at retrieval time, web search engines perform in a different way as the classical IR systems. Current world famous search engines have developed extraordinary formulas to place the most useful web pages on top of the results set. *Page Rank* [14], the core of Google, is one of these world famous formulas for web retrieval. Instead of calculating the relevance of a web page considering only the content of the pages, Page Rank also adds to the equation the external links that reference those pages. A wide IR community is working in improving the current ranking formulas to achieve the best document ranking.

Nevertheless, apart from the quality of the results, a web search engine must answer the queries with sub-second response times. Users have the freedom of choosing their web searcher, and the speed in finding their answers is a key point to be considered. Next section is in charge of introducing how these large-scale search engines can serve thousands of queries in less than a second.

2.4 Distributed Information Retrieval

To answer queries with sub-second response times, large Information Retrieval systems such as Web search engines typically deploy distributed architectures [35]. In such architectures, when new queries arrive at a *broker*, it broadcasts them to the query servers, that contain a subindex of the collection, before collating and merging the results and producing the final top K retrieved set for presentation to the user.

There are two basic ways of partitioning an index into several shards:

- Term Partitioning: each partition holds a subset of the global vocabulary. Therefore, each query server contains all the information related to the terms present in its vocabulary subset. This Document Partitioning approach can also be found in the literature as *System Index* [91] or *Global Index* [83] organization. When a new query arrives, the broker must split and send their terms to the query server containing the information related to each of them. In a document partitioning organization with k parts, a query of $|q|$ terms requires $k \cdot |q|$ disk reads, regarding the $|q|$ reads needed by term partitioning. The main disadvantages of this organization are as follows: the heaviest process is carried out by the broker, and this can result in a bottleneck; the query servers send the inverted lists to the broker, and this may cause bandwidth saturation. Term partitioning can also lead to load imbalance, as the distribution of term occurrences in both collections and query streams is highly skewed [91].

- Document Partitioning: this approach assigns a subset of the documents in the collection to each subindex (also called *shard*). Partitioning can be performed by splitting a monolithic index into the designated subsets by document ids. Alternatively, subcollections of the full document collection can be created, and a separate subindex built for each subcollection. Whichever way partitioning is performed, the resulting subindexes are largely autonomous, capable by themselves of answering queries upon the subcollections they manage [96]. This Document Partitioning approach can also be found in the literature as *Host Index* [91] or *Local Index* [83] organization. In this approach, all queries must be sent by the broker to all query servers, as each of them contains a different subset of the collection. The advantages of Document Partitioning are the following: it is easy to build and maintain and queries are processed in a highly parallelized way, resulting in good average response time. The main disadvantage is the following: as each query server process each query in an entirely autonomous way, the use of local statistics can lead to a degradation in retrieval effectiveness, as local statistics may not represent the global ones in the full collection. Alternatively, global statistics can be distributed by the broker to all query servers. Document partitioning will be the organization used in this thesis for index distribution.

To ensure high throughput rates, shards are often replicated, so that one of multiple query servers can provide the results for a single shard [24]. Indeed, with multiple *replicas* of the same shard, more queries can be processed in parallel on identical shard copies, thus reducing the waiting time of incoming queries. In this way, a cluster of machines operating a large IR system is often arranged according to two orthogonal dimensions [35], as shown in Figure 2.3: the first dimension depends on the number of shards, to improve the processing time of single queries, while the second dimension determines the number of replicas of each shard, to improve the query throughput of the whole system.

These large infrastructures achieve great response time and thoughtput, at the cost of generating some negative consequences that will be detailed in next section.

2.5 Green Information Retrieval

Commercial web search engines are expected to process queries under tight response time constraints and be able to operate under heavy query traffic loads. Operating under these conditions requires building a very large infrastructure involving thousands of computers, with corresponding continuous operating costs. In particular,

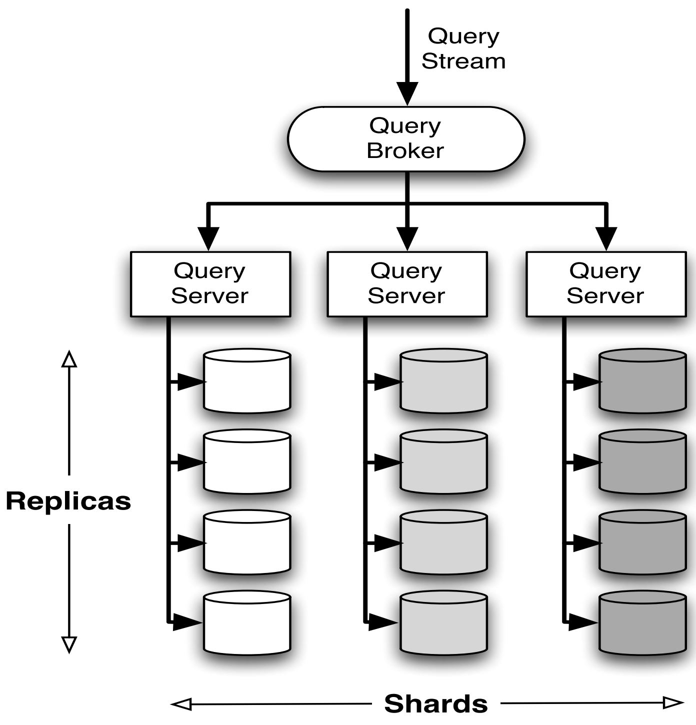


Figure 2.3: Distributed search architecture with shards and replicas.

electricity costs (including power and cooling) form an important part of the operational costs of search engine companies [50]. Indeed, in 2011, Google's overall power consumption was reported[6] to be 2.68 million MWh.

In recent years, the consciousness of environmental problems tied to Green-House Gases has increased. In 2007, analyst Gartner estimated that Information and Communication Technology is responsible for 2% of the total emissions [1]. Much research effort has been oriented to achieve power savings for data centres or Internet servers [38, 55, 59, 74, 95] taking into account different factors such as the devices' power consumption and/or cooling systems. Moreover, large IT companies such as Microsoft[7] and Google[8] are making efforts in reducing their carbon emissions, while also publishing their carbon footprints and goals.

Some directions on how to proceed in order to save power consumption in data centres are given in [59], where Lin et al. showed that the most effective and aggres-

[6]`http://www.google.com/green/`.
[7]`http://www.microsoft.com/environment/`
[8]`http://www.google.com/green/`

sive power saving comes from turning off components that are not used, such as CPU, disk, and memory, which consume substantial power when they are turned on, even with no active workload. Nevertheless, they remark on several challenges regarding this technique: the workload scheduling of the remaining active systems to preserve performance and the cost of waking up a sleeping component.

Chowdhury [32] recently introduced the term *Green IR*, that maps this concern into the Information Retrieval field. His road-map for improving environmental impact of Information Retrieval systems addresses the power efficiency of end-user devices as well as within the search engine itself.

Green behaviour must be adopted by IR systems deployers not only at retrieval process, but also during previous tasks such as evaluation.

2.6 Evaluating an Information Retrieval System

After reading the previous sections, we can conclude that a continuous and ambitious aim in IR research is the improvement of IR systems behavior [25]: search engines compete for satisfying user's search needs and offering the better results to a query. If we want to establish a comparison among different IR systems, we must be able to quantify their quality. Current search engines offer lots of aspects to be considering for user evaluation, as the usability of the system, the simplicity of the interface, the quality and speed of the result set or even the possibility of refine a search. As this thesis is not concerned with physical aspects of the search engine, we will mainly focus in the evaluation of the following factor: *efficiency*.

Evaluation is a major area of research in Information Retrieval. To ensure the repeatability of the IR experiments, researchers re-use shared test collections composed of the following items: a standard document collection; a standard set of queries or topics to be run against that document collection and a standard set of judgments, created by human assessors, that indicates which documents in the collection are relevant to each query. The latter are sometimes known as *qrels*.

This standardization allows to easily reproduce other's experiments and suitably compare results among different studies. Some important test collections are listed bellow [69]:

- The *Text REtrieval Conference (TREC)*[9] consists of a series of workshops covering different IR areas called *tracks*. Each track is provided with the necessary infrastructure (i.e.: datasets, topics) for large-scale evaluation, concerning the topic of the track. These are the most used test collections for IR evaluation.
- The *Cranfield* collection was the first test collection for measuring IR effectiveness (late 1950s) but it size is not large enough to evaluate the current large-scale IR systems.
- *Cross Language Evaluation Forum (CLEF)*: focused on cross-language Information Retrieval.
- *NII Test Collections for IR Systems (NTCIR)*: this is a project that built several test collections of similar sizes to the TREC ones. It is also related to cross-language but mainly focused on East Asian language.

In this thesis GOV2 and ClueWeb09 TREC collections will be used. The former is composed of 25 million pages crawled from the .gov domain to use in the *Terabyte Track*. The latter represents a larger web collection for IR evaluation - 1 billion pages - and it was used by several tracks of the TREC conference. A subset of this collection (ClueWeb09 Category B) has been separated by filtering only the first 50 million English pages.

An important factor to be considered in large-scale IR evaluation is the platform on which the experiments will be carried out. A wide discussion can be made about this topic, so next Chapter will be in charge of discussing the different alternatives we have at our disposal for running our experiments.

[9] `http://trec.nist.gov/`

Chapter 3

Evaluation Platforms

3.1 Introduction

The performance evaluation of large-scale IR systems is challenging. Big companies of web search engines can afford to have many computers (probably thousands) devoted to the development and testing of the IR systems. Unfortunately, this is not the case for most of the research groups. In this case, analytical, simulation or virtualization models are used instead (or even a combination of them [83]). When using a model, independently of its type, one of the first tasks is to compare its performance with a real system in order to verify that both systems present a similar behavior.

This chapter aims to perform a complete study about the most useful large-scale IR evaluation platforms. The main objectives of this study is as follows:

1. On one hand, establishing some recommendations for IR researchers in order to make them easier to choose the most suitable large-scale IR evaluation platform for their necessities.

2. On the other hand, selecting the most suitable IR evaluation platform for the experiments we will perform in the remainder of this thesis.

This chapter is structured as follows: Section 3.2 presents a state-of-the-art review across the different distributed IR evaluation environments, by indicating some published examples over time. Then, in Section 3.3 we compare the different options, by analysing their reliability and cost and detailing their strengths and weaknesses.

Finally, Section 3.4 summarizes the main conclusions, which allow us to choose the most suitable evaluation platform for performing our experiments.

3.2 Background

In this section we analyse the available options to evaluate IR systems, mainly distributed and parallel, in terms of efficiency. We introduce each approach and make a compilation of some works that published their results using the following platforms:

1. Analytical models
2. Simulation
3. Virtualization
4. Cloud Computing
5. Real Systems

3.2.1 Analytical models

This approach consists in developing a mathematical model to represent the behavior of a search engine: queuing theory, average times estimation, etc. Usually some features (i.e.: network delays or seek time for disk) are removed to keep the model simpler and easy to define and manage. However, as the number of modelled features increases, so does the accuracy.

Several works are based on this models, such as [31], where the authors use the queueing network theory to model a distributed search engine. The processing time in a query server is modeled as a function of the number of documents indexed. They build a framework in order to analyse distributed architectures for search engines in terms of response time, throughput and utilization. They also introduced a new cost-based analysis model that finds an optimal set of solutions to consider when constructing a search system.

Another analytical model is used in [83] to study how query performance is affected by the index organization, the network speed, and the disk transfer rate. Their model represents the following retrieval steps: seeking disks, reading inverted lists from disk and processing weights according to the vector model, local ranking of retrieved documents, transferring ranked documents to the central broker, and final ranking at the central broker. This whole retrieval process is replicated for a batch of queries by using a small simulator written in C++. Some of the analytical parameters are checked against a real system.

3.2.2 Simulation

Simulation recreates a set of conditions artificially to represent the search process. It represents more complex behaviors than an analytical model. This is a widely used technique for distributed IR experiments.

A simulation environment called DeNet [60] is used in [91] for comparing the performance impact on query processing of various physical organizations for inverted list. Some simulation parameters as disk bandwidth and I/O settings are extracted from [30]. Lu et al. [61] use simulation to represent a simple peer-to-peer network. Cacheda et al. [23] simulate a distributed IR system using document partitioning by using JavaSim[1], which lies in a simulation package written in Java that allows to represent simulation processes (i.e.: broker, query servers), events (i.e.: query arrivals), queueing algorithms and even statistical routines (you can simulate different query arrival distributions, such as exponential). This platform allows not only the representation of the different components in a distributed and replicated IR system, but also the characteristics of the network that connects them. Another recent approach [71] uses a simulator implemented using C++ and the LibCppSim library [73] for evaluating a new cache hierarchy for web search engines.

3.2.3 Virtualization

Virtualization is a technique that allows the user to run several virtual environments into one physical machine, which is called host. The user can configure each virtual environment on a different way, using different operative systems or setting parameters, quite similar to a real machine. This technology is based on methodologies like hardware and software partitioning, partial or complete machine simulation and

[1] `http://javasim.codehaus.org`

others [11]. Some well-known commercial PC emulators are KVM[2], VMware[3], VirtualBox[4] or Virtual PC[5].

Typical virtualization scheme is shown in Figure 3.1, which shows how the virtual servers that represent the physical ones are hosted into a more powerful machine.

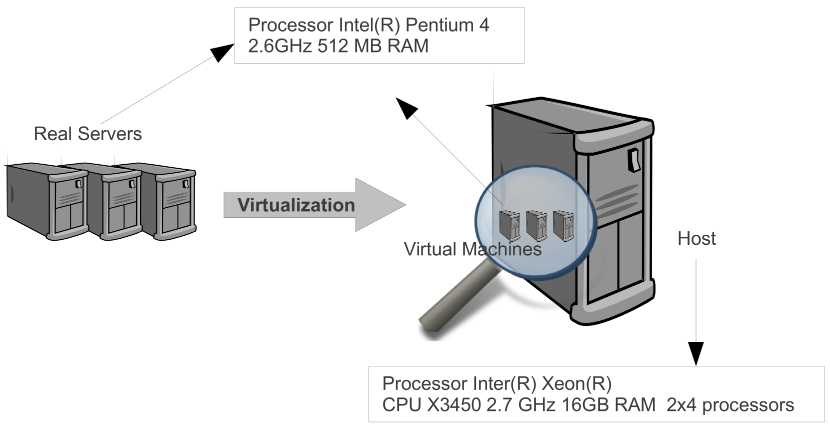

Processor Intel(R) Pentium 4
2.6GHz 512 MB RAM
Real Servers
Virtualization
Virtual Machines
Host
Processor Inter(R) Xeon(R)
CPU X3450 2.7 GHz 16GB RAM 2x4 processors

Figure 3.1: Mapping between real and virtualized architectures.

3.2.4 Cloud Computing

The term *Cloud Computing* has grown in interest. It refers to a modern approach that gives users access to virtual servers, by avoiding the maintenance of physical hardware and reducing the cost and development process. These services are broadly divided into several categories:

- SaaS (Software as a Service): gives the user access to software and its related data hosted on the cloud.
- STaaS (STorage as a Service): cloud approach that works as a remote disk where the user can store his data (i.e.: Dropbox[6] or Google Drive[7]).
- PaaS (Platform as a Service): gives user a deployment environment for creating software by using the provider's tools.

[2]http://http://www.linux-kvm.org/
[3]http://www.vmware.com/
[4]http://www.virtualbox.org/
[5]http://www.microsoft.com/windows/virtual-pc/
[6]http://www.dropbox.com
[7]http://drive.google.com

- IaaS (Infrastructure as a Service): the user takes control over the cloud, since he manages all the resources (sometimes also physical resources), installs his own operative systems and applications, manage the storage system and the network.

Cloud computing services are sold on demand, typically by the minute or the hour. It is elastic, in the sense that a user can have as much or as little of a service as they want at any given time. The services are fully managed by the provider, so the consumer needs nothing but a personal computer and Internet access.

Internet offers an increasing number of cloud computing solutions as Amazon Elastic Compute Cloud (Amazon EC2)[8], IBM Smart Cloud[9], Google Compute Engine[10] and HP Cloud Compute[11].

Authors of [81] present a set of techonologies to run large scale distributed IR systems using both virtualization and cloud computing. This work consitutes an important step forward in using Grid infrastructures for IR purposes.

3.2.5 Real systems

An example of this kind of systems is [7], where authors implement a real distributed architecture and compare the impact of global vs. local partitioning on system performance. A more recent approach [70] describes and evaluates an efficient method for performing parallel query processing upon distributed inverted files. The experiments were performed on a cluster with dual processors (2.8 GHz) that uses NFS mounted directories. This system has 2 racks of 6 shelves each with 10 blades to achieve 120 processors. In [8], the authors investigate and analyse the (im)balance among homogeneous index servers in a cluster for parallel query processing. They configure the cluster with 2 and 7 local inverted indexes.

[8]`http://aws.amazon.com/ec2/`
[9]`http://www.ibm.com/cloud-computing`
[10]`http://cloud.google.com/products/compute-engine`
[11]`http://www.hpcloud.com`

3.3 Analysis

This section is in charge of studying in depth the different evaluation platforms presented in previous section. Several factors must be considered to establish a comparison among the different options we have, and each subsection handles one of them. This way, Section 3.3.1 analyses the use of the different platforms across time. Section 3.3.2 aims to conclude which option is more faithful to a real platform. Financial costs are analysed in Section 3.3.3 and a summary of the strengths and weaknesses is presented in Section 3.3.4.

3.3.1 Chronological analysis

In order to study the chronological evaluation of the IR evaluation platforms, we have collected some published works that specify what platforms they use. Table 3.1 shows interesting data: the platform used (*Type*), the year of publication (*Year*), how many computers they use (*UH*) and how many they represent (*MEH*) (in real systems, *UH*=*MEH*). Our conclusions are as follows:

- Regarding Analytical approach, few examples of these platforms were found, and their use is not recent (encountered examples were performed in 1998 and 2003).

- Simulation is a widely used option across time. The first studies we have found using this approach come from 1990, and we can also find many examples nowadays.

- The architectures with the maximum number of machines were implemented by simulation: 2048 machines in [71] and 1024 machines in [23] and [26].

- Real Systems limit the scale of the experiments (most of them use less than 8 machines), unless in case of big companies with high amount of resources [70].

- Also in case of big companies, they use to build their systems upon simulation platforms, such as in [71] and [6].

Table 3.1: IR evaluation environments used across the time. UH = Number of hosts used for evaluation; MEH = Maximum number of evaluated hosts. (Blanks = UH is not indicated in the paper).

Paper	Year	Type	UH	MEH
[21]	1990	Simulation		16
[91]	1993	Simulation		16
[51]	1993	Simulation		30
[27]	1997	Simulation		128
[83]	1998	Analytical		64
[62]	2000	Simulation		32
[68]	2000	Real	8	8
[7]	2001	Real	5	5
[31]	2003	Analytical		24
[26]	2005	Simulation	1	1024
[75]	2006	Real	8	8
[76]	2007	Real	8	8
[8]	2007	Real	7	7
[23]	2007	Simulation	1	1024
[70]	2007	Real	120	120
[71]	2010	Simulation		2048
[6]	2012	Simulation		512

3.3.2 Real behaviour modelling

The evaluation platforms presented in Section 3.2 fit the real systems behaviour to a greater or lesser extent. This section aims to study the level of accuracy of analytical, simulation and virtualization models (in the case of cloud computing, if we chose IAAS approach, it would behave as a real system; otherwise, it can be considered a virtualization platform).

Analytical approaches used to be the least reliable solutions, due to the difficulty of mapping all the features of a real system into an analytical model. Each model is inherently dependent on the system we want to represent.

With regard to simulation models, Cacheda et al. [23] designed a simulation platform to represent a distributed IR architecture and they clearly accepted the equivalence between both systems (with p-values higher than 0.889 – A high p-value means that there is no statistically significant difference between the two populations –).

Usually, virtualization platforms are thought to reproduce closely the behaviour of a real system. However, to the best of our knowledge, there are not previous studies in charge of attesting the level of accuracy of the virtualization platforms, so we decided to perform our own tests to validate this behaviour.

3.3.2.1 Studying the suitability of virtualization models for performing large-scale IR evaluation

These experiments allowed us to know not only the different parameters that must be taken into account for achieving similar performance than in real systems, but also the limitations of the virtualization platforms. The experimental setup consisted on the following characteristics:

- We developed a real distributed IR system, and afterwards, the same system was implemented in Kernel-based Virtual Machine (KVM) [57].
- The architecture we employed consisted of *IR-Components* [63] for the distribution of brokers and query servers and Managing Gigabytes for Java [40] for query solving.
- Both physical and virtual experiments use the same topology (see Figure 3.2). It consists of a broker which manages several query servers. The number of these query servers is the variable factor in this topology: 2, 3, 5.
- In the real environment, each query server has the following characteristics: Processor Intel(R) Pentium 4 with 2.6 GHz, 512 MB RAM and Ubuntu Server Operating System. We did not use high performance resources, as the chosen data set was smaller than the ones used in real systems. This way, we selected machines with features according to our work load, which is a common practice in IR research [8].
- In the virtualized environment, the guests were configured as the real computers. The main specifications of the host are the following: Processor Inter(R) Xeon(R) CPU X3450 2.7 GHz with 16GB RAM and 2x4 processors.
- The dataset used for the experiments was GOV2 Test Collection from the *Terabyte Track*.

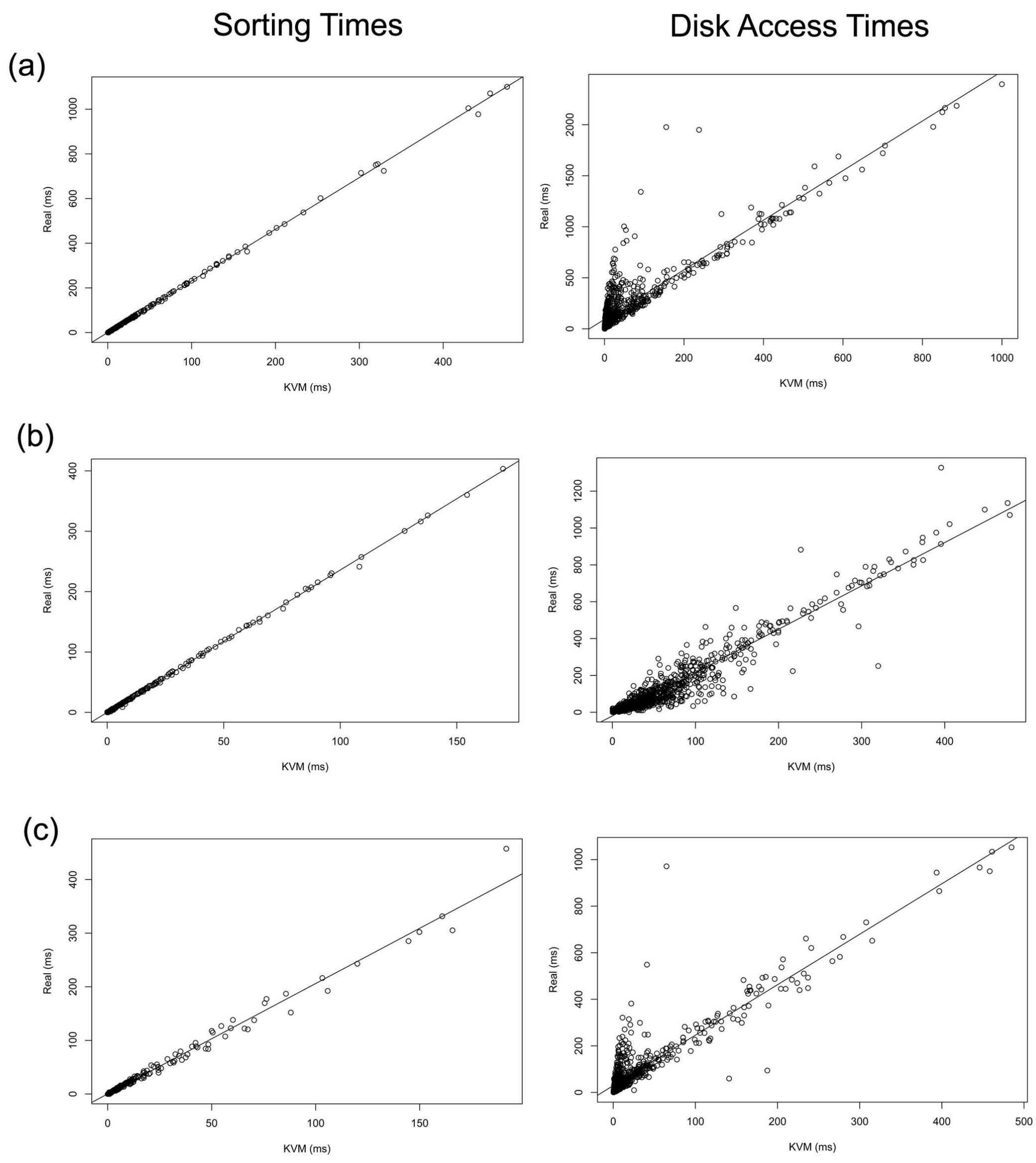


Figure 3.2: Sorting and Disk Access Times for different number of Query Servers: 2 (a), 3 (b) and 5 (c).

The first experiment dealt with the query servers sorting times. We compared the real and virtual times, as it can be seen in Figure 3.2. Outliers were removed from the study. A linear correlation can be clearly established, that explains the 98% of the variability (correlation coefficient (R^2) over 98%). It presents a slope around 2.3, being the virtual query servers faster than the real ones. This is reasonable by considering that the virtual host has more processing capacity than the real query servers. However, as the number of query servers increases, the difference between the real and virtual times is reduced (and so is the slope), due to the higher load of the host. This factor is a limitation of virtualization platforms: as the number of query servers is increased, the host resources for each machine are reduced, reaching a point where they are exhausted and the virtualized system begins to perform worse than the real one. It would be very interesting to find this threshold (number of query servers) and test other approaches for solving this drawback, as the one proposed in [22]. They suggest the building of a scaled-down version of a search engine using virtualization tools in order to create a real distributed system. Following this approach, scaling-down a distributed IR system would maintain the behavior of the whole system and allow the saving of computer resources.

The second experiment is regarding the disk access time of the query servers. In case of 3 and 5 query servers, it can be seen clearly the typical behavior of systems with disk cache (some values are moved away from the main linear correlation). The host maintains a disk cache that improves the response time in the virtual query servers. This problem can be avoided by disabling the disk cache in the host, but the chosen virtualization tool did not allow us to configure the cache. Future experiments include using a more complete virtualization tool to get an even better correlation.

3.3.3 Cost comparison

A crucial factor for deciding our evaluation platform is the cost it will incur in order to finish the experimentation process under a fixed budget. Some aspects to be considered regarding financial costs are the following:

- Obviously, the construction of an analytical or simulation model only concerns the time spent and the machine used to construct the model. This way, these simulation and analytical platforms constitute the cheapest options. Besides, they contribute to the Green IR behaviour, by avoiding the use of many real servers.

- The cost of using virtualization software on a single machine is determined by the virtualization tool, as it can be commercial or free software. Some companies offer free trials or low-functional software for free. This way, we should only consider the number of hosts and their power consumption.

- The cost of cloud computing services changes depending on the hosting company. These companies consider different factors to make the budget, as the user location, the kind of physical resources the user want to use (RAM, CPU...), the number of computing hours, or the quantity of transferred data. While Amazon EC2 is targeting technology-reliant businesses that are turning to the cloud to host their websites, databases and storage, Google is focused initially on research and development teams that may have a need for high-performance computing. The strategy is seen in the pricing models: Amazon EC2 offers reserved instance pricing discounts, in which customers agree to use a compute instance for months or even years. Google's cloud is priced by smaller time chunks and therefore aimed at shorter-lived projects. Once the user has decided to use cloud computing to develop his experiments, there are several web pages [2, 3] that compare the existing cloud computing platforms and they can help the user to choose the most suitable host depending on his research and budget.

- The cost related to a real environment is directly proportional to the number of machines of our architecture. Factors as power consumption and air conditioning must be considered apart from hardware costs.

3.3.4 Strengths and weaknesses

In order to find the best scenario for the evaluation, in terms of efficiency of IR systems, we summarize in this section the advantages and drawbacks of the approaches described previously:

- Regarding the analytical model, its main drawback is that it cannot represent all the characteristics of a real IR system. Although this approach offers high savings, it usually involves the development of complex models and some features have to be dropped to keep the model simple and easy to implement.

- Using a simulation model, we can represent more complex behaviors than with an analytical one. For example, instead of assuming a fixed transfer time for the network, we can simulate its behavior (e.g. we could detect a network saturation). Besides, some works have attested the high reliability of these models if they are designing in detail [23, 26]. That is maybe the reason why simulation has been widely used across time, non only in academical context or small research labs, but also in big companies with enough resources. Besides, this approach, together with the analytical one, are the two platforms that respect the most the Green IR behaviour.

- Virtualization machines can be configured with really similar settings as real ones, but we must be careful with phenomena such as caching and network settings, that may cause unreliable results (see Section 3.3.2). Besides, previous experiments we performed, comparing real vs. virtual distributed IR systems, showed another limitation of virtualization platforms: as the number of virtual query servers launched on the host is increased, the host resources for each of them are reduced, reaching a point where they are exhausted and the virtualized system begins to perform worse than the real one.

- If we focus on cloud computing virtualization, some works [72] identify the strengths, weaknesses, opportunities and threats for the cloud computing industry. As strengths of cloud computing we can emphasize the saving of resources by the user. This also allows the time saving with regard to the hardware maintenance an software updating, as this tasks are usually performed by the host company. The main disadvantage regarding cloud computing is the risk of storing your data outside of your company, on external servers. Besides, this demands high security on data interchanging, which is not always satisfied.

- The reliability of real systems is indisputable; however, if we have not enough resources, some problems can be hidden (i.e.: network saturation) and the systems may have low capacity for big data collections.

3.4 Conclusions

After analysing analytical, simulation, virtualization, cloud computing and real solutions considering different aspects, we summarize our conclusions in table 3.2.

Table 3.2: IR Evaluation techniques comparison

Approach	Finaltial Cost	Setup Difficulty	Reliability
Analytical models	Low	High	Low
Simulation models	Low	High	Medium
Virtualization	Medium	High	Medium
Cloud Computing	Medium	Low	Medium/High
Real Systems	High	Low	High

Although Cloud Computing seems to be a good candidate to perform large-scale IR systems, its problems relying on security data interchange make it a controversial approach. Besides, cloud computing servers usually share their resources among different tasks and users, and this can alter their results. Virtualization is not a widely used tool for evaluation, maybe due to the difficulty for controlling some crucial parameters or just due to expensive licences in some specifical distributions. Nevertheless, simulation is a really economical approach used largely for IR evaluation, also for big companies, as we could see in table 3.1. This reason, the lack of real resources, the fact that some works [23] had developed a high reliable simulation environment for distributed IR architectures and the desired Green IR behaviour, make us to chose simulation for the evaluation processes of this thesis.

Chapter 4

Query Scheduling using Prediction

4.1 Introduction

As it was introduced in Chapter 2 a distributed and replicated information retrieval system consists of several query servers, each of them storing a subset of the collection, and several replicas for each query server in order to increase the throughput of the whole system.

When the broker of a distributed IR system receives a query, it must obtain results for that query from each shard, before returning the merged list to the user. Hence, for each shard, it must select the replica that will process the query. The selected replica queues the query until it is ready to process it and return the results to the broker. There are advantages to the user experience[1] and overall system throughput if each query is *scheduled* such that the time it spends waiting in a replica's queue is minimized. Hence, replica selection is carried out by a *scheduling method*, such that the replica selected will permit the query to be answered in as little time as possible.

The problem tackled in this chapter is how a broker should select (*schedule*) the most suitable replica of a given shard in order to reduce the queue waiting time. For example, the replica with the minimum number of queued queries can be selected.

However, the response time for different queries can vary widely, particularly if dynamic pruning is employed [92], such as WAND [17], which aims to avoid the scoring of postings for documents that cannot make the top K retrieved set. Hence, the

[1]Indeed, users exhibit preferences for faster search engines [18].

accurate choice of replica is made more difficult, as the number of queries queued by a given query server does not accurately predict the processing backlog of the server. A recently proposed technique for *query efficiency prediction* [92] offers a plausible manner to estimate the workload of a replica. Hence, we hypothesise that query efficiency prediction [67] can permit accurate query scheduling in a distributed/replicated IR system. Indeed, to the best of our knowledge, there are no other studies that apply query efficiency predictors for scheduling in a distributed/replicated IR system.

4.2 Background

4.2.1 Scheduling

Historically, the scheduling of queries to replicas has not seen much examination in the IR literature. Here, we show three scheduling methods that can be adopted to select the replica for a new query:

- **Random (RD)**: the replica is chosen randomly.
- **Round Robin (RR)**: aims to balance the query traffic across the available replicas. In particular, modulo the number of replicas, if replica i was selected for the previous query, replica $i + 1$ is used for the current query.
- **Queue Length (QL)**: schedules the query to the replica with the minimum number of queries waiting. However, as queries have different response times, one query may be held up behind a slow query, when another replica with shorter queries was available.

Such simple scheduling methods minimize queueing time only when each query has an equal response time [58]. But this is clearly far from reality: all search engines are based on the inverted index data structure [35], which permits the efficient lookup of all documents that contain occurrences of each term in a query. The traversal of postings lists in the inverted index represents a large contribution to the time for a search engine to retrieve documents in response to a query. Indeed, Moffat et al. [76] stated that the response time of a query is related to the posting list lengths of its constituent query terms.

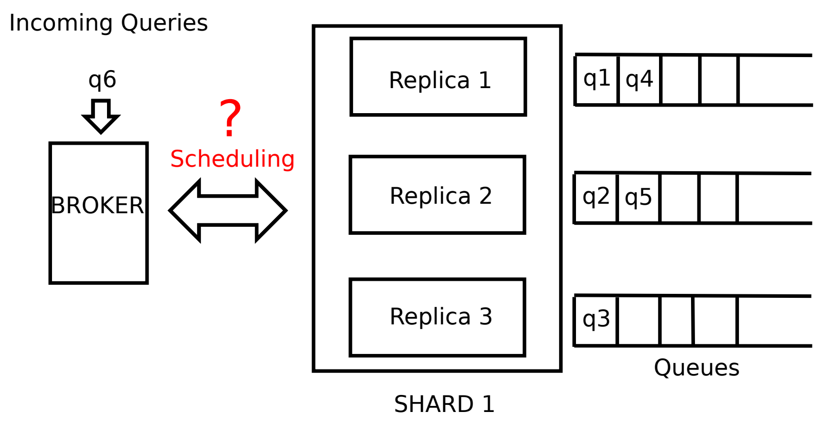


Figure 4.1: Query scheduling, where the Broker must select one replica of a Query Server (shard) to send the incoming query.

Figure 4.1 represents a scenario where a Broker must select one replica of a query server for processing the new query $q6$. This decision is made based on the scheduling method used. In example:

- Random would select a random replica among $Replica1$, $Replica2$ and $Replica3$.
- Round Robin would select $Replica3$, as $q5$ was scheduled to $Replica2$. We assume that Round Robin was the method used to scheduled the queries presented in the Figure.
- Queue Length would also select $Replica3$, as that is the low loaded replica.

If we assume the next query processing times (PT) of table 4.1 for the enqueued queries, it is easy to see that: $PT(q3) > PT(q1) + PT(q4)$ and also $PT(q3) > PT(q2) + PT(q5)$. The fastest queue would be $Replica2$, with a waiting time around $PT(q2) + PT(q5) = 4ms$.

Query	Processing Time
q1	2 ms
q2	3 ms
q3	6 ms
q4	3 ms
q5	1 ms

Table 4.1: Processing time of queries of Figure 4.1

4.2.2 Dynamic Prunning

As we stated in previous section, not all the queries submitted to a search engine take the same time to complete. The number of postings scored has a high impact on the retrieval time. *Dynamic Pruning Strategies* improve efficiency by shortcutting or omitting the scoring of the postings of documents that will not be retrieved in the top k documents [92]. Such postings are said to have been *pruned*. All state-of-the-art works [17, 80, 93] avoid scoring parts of the posting lists, to save disk access, decompression and score computation cost. This mechanism is implemented by maintaining additional information during retrieval: a *threshold* that represents the minimum score that documents must achieve to have a chance to be present in the final top *k* results; and a *term upper bound*, for each query term, that represents the maximal contribution of that particular term to any document score.

While some techniques are based on the pre-sorting of inverted index posting lists by the impact (e.g. [5]), in this work, we focus on the WAND *dynamic pruning* strategies where the inverted index posting lists remain in ascending document-identifier order, as deployed by at least one major search engine [35].

In the WAND dynamic pruning strategy, the posting lists of all query terms are processed in parallel, in a document-at-a-time fashion, such that all postings for a given document are scored before processing moves onto the next document. WAND works by repeatedly calculating a *pivot term*, which the next document to be fully scored must contain. The next document containing the pivot term is the *pivot document* – only if the document contains sufficient query terms to be retrieved in the top K will it be scored. A major benefit of WAND is that it can use skipping [77] forward in posting lists, which reduces posting list decompression overheads, and can reduce IO, with resulting improvements in efficiency [39, 66].

4.2.3 Query Efficiency Predictors

The notion of *predicting query difficulty* refers to techniques that infer the (effectiveness) performance of a given query, without knowing the relevance assessment information [92].

For WAND, the length of the posting lists has been shown to be insufficient to accurately predict the response time of a query [67]. In fact, the response time of WAND depends also on the number of postings that are actually scored, as well as the *pruning difficulty* of the query, i.e. the number of postings that overlap for the constituent query terms, and the extent to which high-scoring documents occur towards the start of the posting lists. Tonelloto et al. [92] proposes the notion of *query efficiency predictors* for estimating the responste time of a query. Various term-level statistics are computed for each term off-line. These are pre-retrieval predictors and have the advantage of allowing changes in the retrieval strategy before retrieval starts [47]. When a new query arrives, the term-level features (i.e. frequency, number of postings, Inverse Document Frequency – IDF) are aggregated into query-level statistics, which are used as input to a learned regression model. The regression model can then produce accurate response time estimations for unseen queries.

Several works have attested the power of these query efficiency predictors. Daniele et al. [15] use query efficiency predictors to feed a load-sensitive selective pruning framework and they also demonstrate that a mutiple feature predictor using DAAT is more accurate than a single feature one. In [16], authors use predictors to introduce a novel dropping strategy for maintaining the response times under a specified threshold.

4.3 Proposal

We have studied on the previous section the problem that arise with the most useful query scheduling techniques. The main contribution of this section is to present a new query scheduling method named **Least Loaded (LL)**, that addresses the inefficiencies of previous approaches, by estimating the workload of a replica more accurately than simply the length of its queue. Using a simulated distributed/replicated search environment, based on actual query response times, we experiment to determine how this new proposal performs for replica selection in comparison with existing scheduling techniques.

We hypothesise that using predicted response times can increase overall efficiency compared to other scheduling algorithms. We propose a new scheduling method, Least Loaded, that sums the predicted response times of the queued queries for each replica. The least loaded replica in terms of predicted availability is selected. Next sections are in charge of presenting the experimental setup and results obtained by comparing our proposal with state-of-the-art scheduling methods.

4.4 Experimental Setup

4.4.1 Datasets

To address our hypothesis, we must obtain the actual and predicted processing times of a set of queries in order to feed later the simulation platform and perform the experiments. With this purpose, we index TREC GOV2 corpus using Terrier[2], applying Porter's English stemmer and removing standard stopwords. For this task we use a quad-core Intel Xeon 2.4GHz, with 8GB RAM, with inverted indices are stored on a 160GB SATA drive.

We conduct experiments by defining two different query sets:

- **Query Set A**: constitutes the first batch of experiments. For retrieval on each query server, we use a set of 2200 queries of the TREC 2005 Terabyte track Efficiency task. We sample real arrival times of a set of queries from an Excite query log and assign them to our TREC queries. We perform the training on a separate subset of 2500 Efficiency task queries.

- **Query Set B**: Best practices in efficiency experiments demand a large number of queries, however the number of queries used can vary largely, from the 50-150 TREC topics used in [94] to thousands of queries from commercial search engines used in [28]. Besides, the volume of queries experienced by a search engine changes dynamically throughout the day[3], with busy periods during the daytime, quiet periods early in the morning and at night, and periods of rising and falling traffic in between [88]. The search engine must be provisioned with sufficient shards and replicas such that the desired level of efficiency can be attained at peak periods. However, during quiet periods there is excess capacity within the search engine, with no contention for replicas, and hence no waiting

[2]`http://terrier.org`

[3]Search data centres are typically geographically distributed, and hence a data centre is likely serving traffic mostly originating from its continent.

time for queries. That is the reason why we decided to use a larger sample of consecutive user queries from a publicly available real search engine log, thereby measuring the mean query response time for retrieval. In particular, we select two batches of 10,000 consecutive queries from the MSN 2006 query log [34]. Figure 4.2 shows the query arrivals in a day of activity (sampling rate 120 seconds) and the two representative batches selected for experiment. The first batch (called *Set 1*) contains 10,000 queries in a low traffic time window, i.e. large inter-arrival times, running from 00:00 to 03:30 roughly. The second batch (called *Set 2*) contains 10,000 queries in an high traffic time window, i.e. small inter-arrival times, running from 11:00 to 11:15 roughly. Moreover, these queries exhibit all of the expected properties of a query log, such as frequently repeated 'head' queries and a tail of infrequent queries. For training/testing purposes, each query set is split in two chronologically, half for training, and half for testing. After studying the query arrival times of both sets of queries, we could know that low frequency query arrival time distribution fits Poisson (with mean=0.75 queries/second) and high frequency distribution fits Poisson (with mean=11.5 queries/second). With the aim of comparing the behaviour of the system facing each set of queries with interchanged arrival frequencies, we generated a Poisson distribution with mean=0.75 queries/second and we assigned those arrival times to *Set 2* in order to simulate low query traffic. In the same way, we generated a Poisson distribution with mean=11.5 queries/second for simulating high query traffic in *Set 1*. In order to take into account an intermediate scenario, we also simulate a medium frequency query arrival time distribution following a Poisson with mean=0.5 queries/second and we use this distribution for setting query arrival times in both sets.

In order to calculate the processing times of queries, we perform the retrieval process of all sets of queries. We apply WAND [17] dynamic pruning strategy, which selects $K = 1000$ documents, where each document has been scored for each query using the parameter-free DPH Divergence from Randomness weighting model [4]. DPH is a parameter-free model, which exhibits similar effectiveness to BM25 [85].

As we experiment with three different index configurations (i.e. 2, 5 and 10 shards) we perform the indexing and retrieval process three times in order to get the

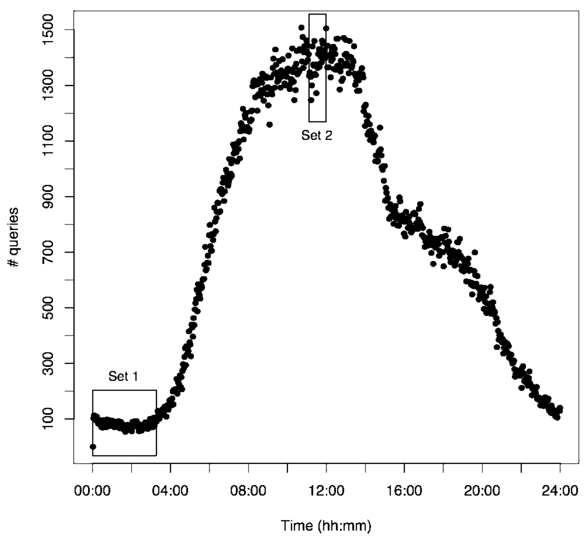


Figure 4.2: Query distribution in a 24 hours time span, with batches selected for experiments.

actual and predicted processing times of queries for all settings.

4.4.2 Predictors

As we explain on previous sections, we obtain the response time prediction following Tonellotto et al. [92], by calculating various term-level statistics and its aggregations to form a total of 113 features. Table 4.2 shows the statistics used. From this table we highlight some representative term statistics:

- *# Postings*: the number of postings in a term's posting list.
- *Maxima*: a term that has fewer maxima in the score distribution may be easier to prune.
- *Promotions into k*: if this query term was the only query term, how many documents containing this term would make it into the top *k* retrieved. A term with a low number of promotions probably has its highest value documents towards the start of the posting list.

- *IDF* (Inverse Document Frequency): describes the informative amount that a term carries.
- *AvICTF* [48]: Average *ICTF* (Inverse Collection Term Frequency). *ICTF* is like the *IDF* except that it also accounts for term frequencies, not just binay.
- $\gamma 1$ [48]: the standard deviation of the *IDF* of the terms in the query.
- $\gamma 2$ [48]: the quotient between the maximum and minimum *IDF* among the terms in the query.
- *SCQ [97]*: the Similarity score between the Query and the Collection.

Predictor	#
Predictors from Term-Based Statistics	
Arithmetic, geometric, harmonic means of score	24
Max Score	8
Approximation of max score	8
Variance of score	8
# Postings	8
# Maxima	8
# Maxima > avg score	8
# Postings with max score	8
# Postings with 5% of max score	8
# Postings with score within 5% of final k threshold	8
Promotions into k	8
IDF	8
Query Performance (Effectiveness) Predictors	
AvICTF [48]	1
AvIDF [48]	1
$\gamma 1, \gamma 2$ [48]	2
SCQ [97]	1
TOTAL	113

Table 4.2: All tested query effectiveness predictors. Term-based statistics are aggregated into efficiency predictors using 8 different functions: sum, max, min, mean, median, range, variance and standard deviation.

Some performance (effectiveness) predictors are also used for indicating efficiency as they attempt to measure how well covered the query is in the corpus. Predicted response times are obtained by gradient boosted regression trees [44] - we use the Jforests implementation of gradient boosted regression trees [46][4].

[4]`http://code.google.com/p/jforests/`

The good performance of multiple feature predictors with regard to a simple feature one has been attested by several works ([92, 15]), and they have encouraged us to use the combination of all the previous statistics.

4.4.3 Comparable Algorithms

In order to test the performance of our algorithm, we also implement the three scheduling methods presented in section 4.2.1: Random, Round Robin and Queue Length.

As the selection of replicas is based on predicted response times, we additionally implement an Oracle scheduling algorithm, which knows the actual response time of a query before it is executed, but still accounts for calculating the predicted response time. This way, Oracle represents a best-case scenario for Least Loaded scheduling. Finally, to compare the five scheduling algorithms, we use two measures: Average Waiting Time (AWT) and Average Completion Time (ACT) over all the queries, in milliseconds (ms). Note that the Average Completion Time is inclusive of the Average Waiting Time. Some tables also include the 90th Percentile (the value below which the 90% of the observations can be found, denoted as 90thPC).

4.4.4 Simulation Setup

In comparing the different scheduling algorithms, we experiment with a various numbers of shards and replicas. To facilitate such experiments without exhaustive hardware resources, we build a simulation framework that supports different distributed settings. Indeed, as we concluded in Chapter 3, a simulation framework can accurately model the efficiency of a real distributed IR system, including the network delays, the queue waiting and processing time for queries and the time for merging the results.

The simulation framework defined in this work represents a distributed IR system encapsulating the roles of a single query broker and multiple query servers, with a local area network interconnect. This is constructed following the simulation framework described by Cacheda et al. [23], with the addition of replicas, and appropriate query scheduling. Our simulation framework is implemented on top of the JavaSim[5]

[5] `http://javasim.codehaus.org`

platform.

Input to the simulation framework takes the form of a stream of queries, with a corresponding arrival time, as well as, for each shard, the predicted and actual response times observed from a real IR system.

When the broker sends the query to the selected replica of each shard, there is a network delay (in secs) that depends on the query size q_l (in MB):

$$L_{oh} + \frac{q_l}{L_{bw}} \tag{4.1}$$

where L_{oh} refers to the network overhead (in secs) for each packet that has been sent and L_{bw} represents the network speed (in MB/s). The values used in [23] are assumed for the network conditions: network delay L_{oh} = 0.1ms, network speed L_{bw} = 100 Mbps = 12.5 MB/s. The return of the $K = 1000$ document results from a query server is K_l = 8KB in size (consisting of one integer and one single float for each result).

Each replicated query server maintains a queue of queries. This queue is processed by waiting for the actual response time observed for the real IR system for that query on that shard. Finally, when the query server returns the results to the broker, a network delay (measured in secs) occurs, which is calculated in terms of K_l, i.e. the size, in MB, of the K returned documents.

For the scheduling methods, we assume that Random, Round Robin and Queue Length methods have a negligible processing time. However, in the Least Loaded method, the time required for computing the query predicted response time has to be taken into account. We extracted a linear correlation between the query length q_l and the time spent for calculating the predictors d_{pred} (in ms). This is used for simulating the delay that this scheduling method adds to the system:

$$d_{pred} = 6.50815 \cdot q_l \tag{4.2}$$

Indeed, over the 5,000 training queries, the average time to make an efficiency prediction is 13.52ms.

Replicas	Random		Round Robin		Queue Length		Least Loaded		*Oracle*	
	ACT	AWT	ACT	AWT	ACT	AWT	ACT	AWT	*ACT*	*AWT*
2 Shards										
2	9,617	9,382	10,061	9,826	8,897	8,662	**613**	**362**	*610*	*359*
4	902	667	409	174	434	199	**253**	**3**	*253*	*3*
8	410	175	263	28	428	193	**250**	**0**	*250*	*0*
5 Shards										
2	375	237	241	103	247	109	**158**	**4**	*159*	*5*
4	265	126	155	16	231	93	**154**	**0**	*154*	*0*
8	192	54	**140**	2	231	93	154	**0**	*154*	*0*
10 Shards										
2	168	69	120	22	145	47	**114**	**1**	*114*	*1*
4	139	41	**101**	3	144	46	114	**0**	*114*	*0*
8	123	25	**98**	**0**	144	46	114	**0**	*114*	*0*

Table 4.3: ACTs and AWTs (in milliseconds) for different settings and scheduling algorithms.

4.5 Results

In this section we present the results obtained after running the five scheduling methods introduced on previous sections. We divide our result analysis into two subsections related to Query Set A and Query Set B respectively.

4.5.1 Query Set A

Results regarding Query Set A are shown in Table 4.3, where we present ACT and AWT for all the scheduling algorithms. From Table 4.3, we note that increasing both the number of shards and the number of replicas reduces both ACTs and AWTs. Indeed, in general, 2 shards with only 2 replicas is insufficient for a low completion time for this query workload, as queries can spend 8 seconds waiting for an available query server. For 5 or more shards, more than 4 replicas is sufficient for eliminating any *contention* for query servers (i.e. AWTs close to 0).

In general, a fixed query volume can be serviced by a larger number of replicas for a smaller number of shards, or a large number of shards with a smaller number of replicas. As the number of shards increases, the number of predictions that are required for each query rises. However, as the contact to each query server occurs in parallel to obtain the efficiency predictions, the overhead of increasing the number of shards should be minimized.

Comparing the scheduling algorithms, we note that Random obtains the highest ACTs and AWTs, because it can choose replicas that are busy, whist other replicas for that shard are idle. Queue Length is superior to Round Robin under high contention (i.e. 2 shards, 2 replicas). In other settings, Round Robin appears to better balance load than Queue Length. However, across different numbers of shards and replicas, Least Loaded always achieves the smallest AWT. For instance, with 4 replicas of the 2 shard index, Least Loaded can reduce AWT to 3ms, compared to 199ms for Queue Length and 174ms for Round Robin. Under settings with very little contention (e.g. 10 shards, 4 or 8 replicas), Round Robin has slightly lower ACTs than Least Loaded and even Oracle, due to the expense of predicting the response time (typically 6-40ms, depending on query length). Finally, Least Loaded obtains ACTs and AWTs that are almost identical to the best-case Oracle algorithm, based on actual response times.

Overall, we find that using predicted response times to select the suitable replica for each query results in improved efficiency.

4.5.2 Query Set B

In order to examine the effect of a larger number of queries and different query traffic flows, we study the results using Query Set B. Tables 4.5 - 4.10 (at the end of the chapter) show average completion times (in milliseconds) and 90th Percentile for both sets of queries and different architectures. These experiments also include more number of replicas (from 2 to 10).

We start studying query traffic influence. In order to analyse this factor, we must focus on tables 4.5 - 4.10. Best ACT are in bold, and the second best method is in brackets. Results in tables 4.5 and 4.6 attest that Least Loaded is not always the best method for scheduling. Round Robin achieves better ACT for most configurations. Nevertheless, medium query traffic scenario (Tables 4.7 and 4.8) presents different behaviour: except for one configuration of set 1 (10 shards, 9 replicas) and the configurations of set 2 corresponding to 10 shards and more than 7 replicas, all the configurations perform better by using Least Loaded as the scheduling method. Tables 4.9 and 4.10 highlight the good performance of query efficiency predictors under high query contention scenarios: Least Loaded achieves the best ACT with all the tested architectures. For instance, the configuration Set 1, 2 shards, 10 replicas reduces the ACT by 68% with respect to Queue Length.

To fully investigate this influence factor, we present complementary results in terms of AWT. Figures 4.3 - 4.5 represent the AWT obtained for all the scheduling methods after running query Set 2 under the three query traffic conditions. The difference between Least Loaded and the other methods is higher as the query traffic increases. Graphics for high query traffic show how only Least Loaded (for more than 2 shards) is able to process all the incoming queries under acceptable waiting times.

All these results attest that the higher query traffic is, the best Least Loaded performs comparing to the other scheduling methods. This way, under high contention scenarios, Least Loaded leads to better ACT and AWT.

Table 4.4: Number of replicas for which prediction is worth for different number of shards and query traffic

#Shards	Query Set 1	Query Set 2
Low Query Arrival Times		
2	<6	<4
5	<4	<4
10	<4	<4
Medium Query Arrival Times		
2	always	always
5	always	always
10	always	<8
High Query Arrival Times		
2	always	always
5	always	always
10	always	always

With the aim of summarizing previous results, we present Table 4.4, which recommends the number of replicas that we should use if we want to use prediction without prediction calculation delay penalty, that is, the best configurations for achieving the best performance with Least Loaded method. It is easy to deduce how the architecture affects the results: as the number of shards increases, Least Loaded seems to be a less suitable scheduling method. The same correlation exists regarding the number of replicas. However, Least Loaded is the best option when query traffic is high, independently of the architecture.

In Table 4.4 we can also prove that the behaviour of both query sets is different under different scenarios. If we study in depth the average processing time (APT = ACT-AWT) for each set of queries, we can deduce that Set 1 contains more difficult queries than Set 2: APT(Set1) = 10555ms vs APT(Set2) = 7721ms, considering low query traffic. With medium query arrival rates, the difference decreases: APT(Set1) = 6960ms vs APT(Set2) = 6961ms and for high arrival rates, the APT of both sets is almost the same (difference concerns decimals).

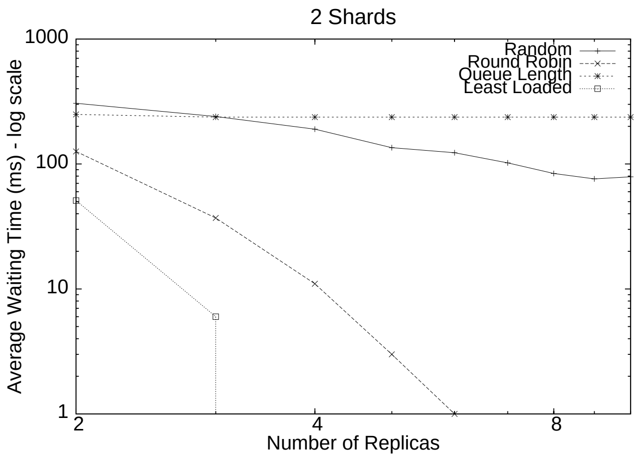
This fact can explain the difference between the behaviour of both query sets: as Set 1 contains more difficult queries than Set 2, prediction method performs better than the other scheduling methods under more configurations (i.e.: medium query arrival rates and 10 shards).

4.6 Conclusions

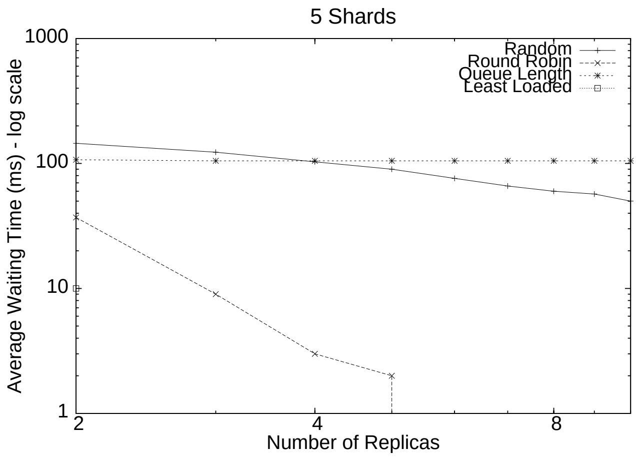
In this chapter we have proposed that using the predicted response time (obtained using query efficiency predictors) could enhance replica selection within a distributed and replicated IR system, compared to other scheduling algorithms.

Indeed, experiments using the GOV2 corpus showed that the proposed Least Loaded algorithm could attain marked reductions in the query waiting times, across different number of shards and replicas. We have performed the experiments using different architectures in order to study the effect of the number of shards and replicas. The experiments were also driven using different query sets. The use of the second query set provides the experiments with a larger set of queries and different query traffic rates. Some arrival rates were generated synthetically, while others were extracted from real query logs. For the first query set, using predicted response time to select between 4 replicas of a 2 shard index results in a 42% reduction in mean completion time compared to selecting replicas by considering only the length of their queues. Nevertheless, when there is low contention in the system, Least Loaded introduces a delay in the system caused by the time spent in predictors calculation.

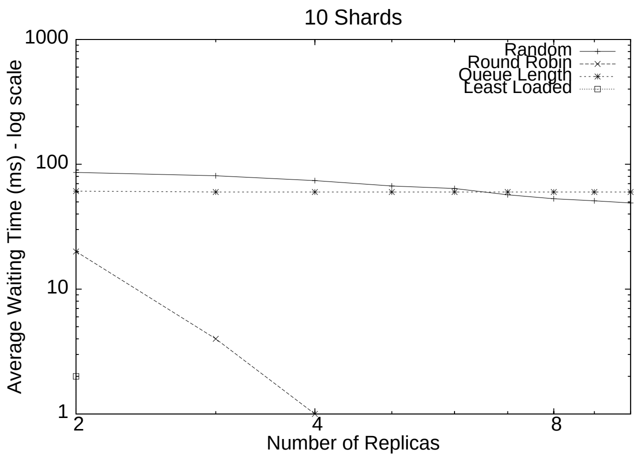
Next chapter analyses this problem in depth and offers a novel solution.



(a)

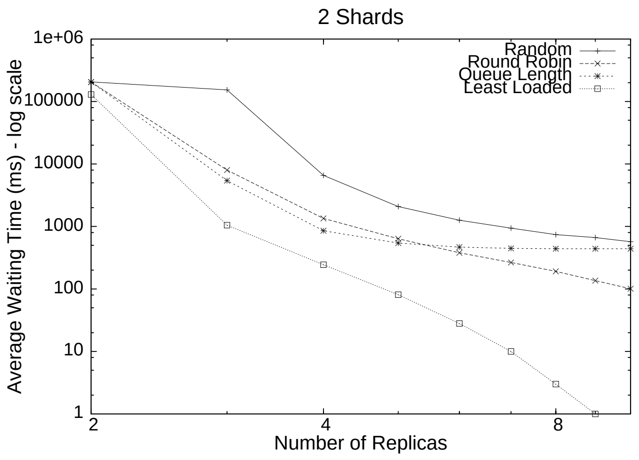


(b)

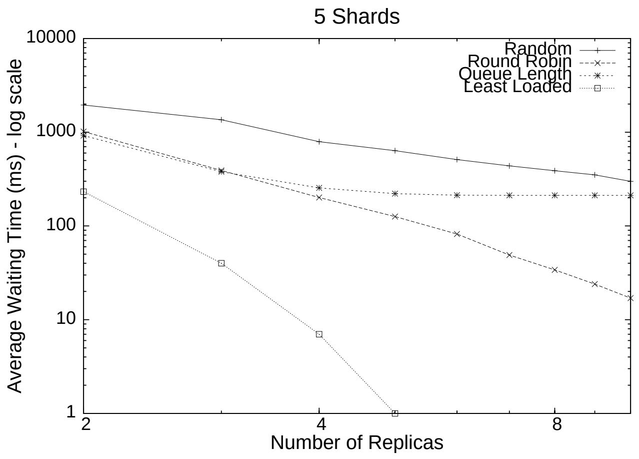


(c)

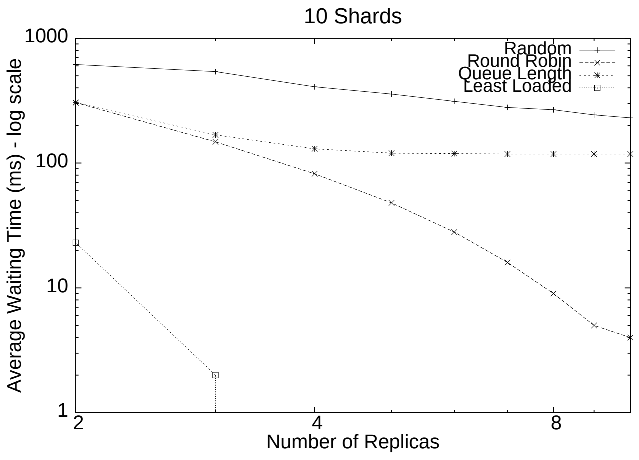
Figure 4.3: Average Waiting Time for different architectures and scheduling methods under **LOW query traffic**



(a)

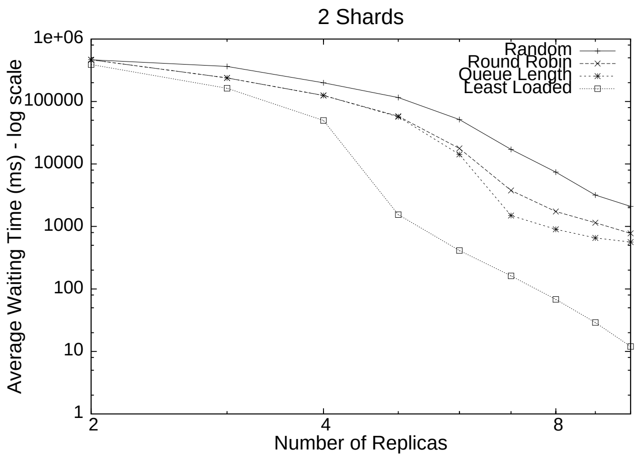


(b)

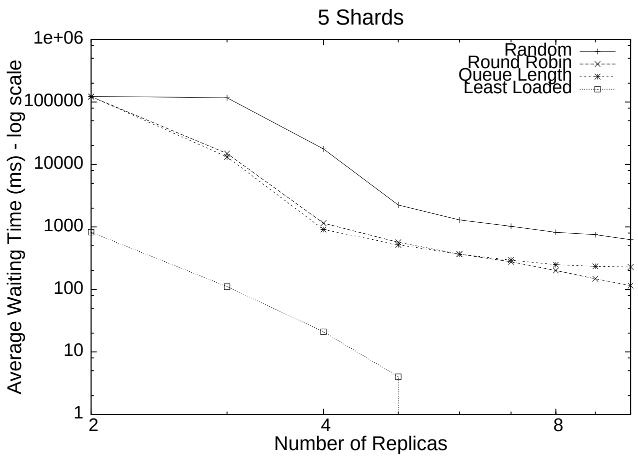


(c)

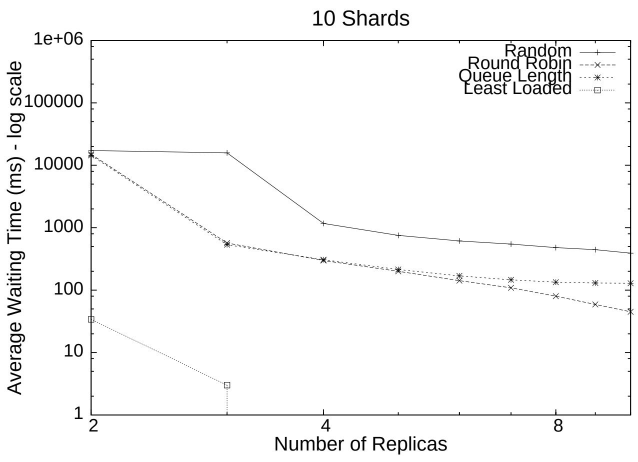
Figure 4.4: Average Waiting Time for different architectures and scheduling methods under **MEDIUM query traffic**



(a)



(b)



(c)

Figure 4.5: Average Waiting Time for different architectures and scheduling methods under **HIGH query traffic**

Table 4.5: Average Completion Times - **LOW** arrival rate - **SET 1**

	SET 1			
R	**LL**	90thPC	**2nd Best**	90thPC
	2 Shards			
2	**922**	2,919	1,096 (RR)	3,280
3	**1,665**	3,711	6,004 (QL)	12,496
4	**798**	2,658	836 (RR)	2,796
5	**696**	1888	1142 (QL)	2,843
6	793	2,657	**790** (RR)	2,651
7	**626**	1,682	865 (RR)	2,463
8	792	2,657	**779** (RR)	2,645
9	**617**	1,666	736 (RR)	2,231
10	792	2,657	**778** (RR)	2,645
	5 Shards			
2	**492**	1,013	553 (RR)	1,415
3	**400**	1,111	724 (QL)	1,624
4	470	1,292	**469** (RR)	1,287
5	**361**	1,082	471 (RR)	1,206
6	469	1,292	**457** (RR)	1,275
7	**360**	1,082	394(RR)	1,120
8	469	1,292	**453** (RR)	1,275
9	**360**	1,082	369 (RR)	1,083
10	469	1,292	**453** (RR)	1,275
	10 Shards			
2	**304**	714	321 (RR)	740
3	**249**	671	380 (RR)	847
4	301	711	**289** (RR)	704
5	**247**	665	279 (RR)	688
6	301	711	**285** (RR)	694
7	247	665	**247** (RR)	676
8	301	711	**285** (RR)	692
9	247	665	**236** (RR)	659
10	301	711	**285** (RR)	692

Table 4.6: Average Completion Times - **LOW** arrival rate - **SET 2**

	SET 2			
R	**LL**	90thPC	**2nd Best**	90thPC
	2 Shards			
2	**667**	1835	726 (RR)	2202
3	**1665**	3711	6004 (QL)	12496
4	616	1668	**611** (RR)	1671
5	**696**	1888	1142 (QL)	2843
6	616	1666	**601** (RR)	1639
7	**626**	1682	865 (RR)	2463
8	616	1666	**600** (RR)	1636
9	**617**	1666	736 (RR)	2231
10	616	1666	**600** (RR)	1636
	5 Shards			
2	**370**	1094	382 (RR)	1121
3	**400**	1111	724 (QL)	1624
4	361	1,082	**348** (RR)	1,067
5	**361**	1082	471 (RR)	1,206
6	361	1,082	**345** (RR)	1,062
7	**360**	1082	394(RR)	1,120
8	361	1,082	**345** (RR)	1,062
9	**360**	1082	369 (RR)	1,083
10	361	1,082	**345** (RR)	1,062
	10 Shards			
2	**249**	670	251 (RR)	684
3	**249**	671	380 (RR)	847
4	247	665	**232** (RR)	649
5	**247**	665	279 (RR)	688
6	247	665	**231** (RR)	646
7	**247**	665	247 (RR)	676
8	247	665	**231** (RR)	646
9	**247**	665	236 (RR)	659
10	247	665	**231** (RR)	646

Table 4.7: Average Completion Times - **MEDIUM** arrival rate - **SET 1**

	SET 1			
R	**LL**	90thPC	**2nd Best**	90thPC
	2 Shards			
2	**261,760**	541,895	340,467 (RR)	672,288
3	**1,665**	3,711	6,004 (QL)	12,496
4	**1,708**	3,934	6,648 (QL)	14,511
5	**696**	1,888	1,142 (QL)	2,843
6	**864**	2,696	1,619 (QL)	3,966
7	**626**	1,682	865 (RR)	2,463
8	**800**	2,658	1,404 (RR)	3,695
9	**617**	1,666	736 (RR)	2,231
10	**792**	2,658	1,129 (RR)	3,238
	5 Shards			
2	**1,170**	3,037	6,082 (QL)	15,225
3	**400**	1,111	724 (QL)	1,624
4	**490**	1,225	954 (QL)	2,347
5	**361**	1,082	471 (RR)	1,206
6	**470**	1,293	729 (RR)	1,935
7	**360**	1,082	394(RR)	1,120
8	**469**	1,293	615 (RR)	1,493
9	**360**	1,082	369 (RR)	1,083
10	**469**	1,293	550 (RR)	1,410
	10 Shards			
2	**357**	774	966 (QL)	2,548
3	**249**	671	380 (RR)	847
4	**301**	711	503 (QL)	1,104
5	**247**	665	279 (RR)	688
6	**301**	711	379 (RR)	874
7	**247**	665	247 (RR)	676
8	**301**	711	335 (RR)	744
9	247	665	**236** (RR)	659
10	**301**	711	312 (RR)	721

Table 4.8: Average Completion Times - **MEDIUM** arrival rate - **SET 2**

	SET 2			
R	**LL**	90thPC	**2nd Best**	90thPC
	2 Shards			
2	**130,316**	236,367	205,329 (QL)	370,826
3	**1,665**	3,711	6,004 (QL)	12,496
4	**859**	2,305	1,455 (QL)	3,309
5	**696**	1,888	1,142 (QL)	2,843
6	**644**	1690	980 (RR)	2,648
7	**626**	1682	865 (RR)	2,463
8	**619**	1671	791 (RR)	2,304
9	**617**	1666	736 (RR)	2,231
10	**616**	1666	701 (RR)	2,136
	5 Shards			
2	**592**	1,354	1,265 (QL)	2,658
3	**400**	1,111	724 (QL)	1,624
4	**368**	1,088	545 (RR)	1,306
5	**361**	1,082	471 (RR)	1,206
6	**360**	1,082	427 (RR)	1,163
7	**360**	1,082	394(RR)	1,120
8	**360**	1,082	379 (RR)	1,101
9	**360**	1,082	369 (RR)	1,083
10	**360**	1,082	362 (RR)	1,079
	10 Shards			
2	**270**	691	536 (QL/RR)	1,144/1,137
3	**249**	671	380 (RR)	847
4	**247**	665	313 (RR)	751
5	**247**	665	279 (RR)	688
6	**247**	665	259 (RR)	684
7	**247**	665	247 (RR)	676
8	247	665	**240** (RR)	667
9	247	665	**236** (RR)	659
10	247	665	**235** (RR)	656

Table 4.9: Average Completion Times - **HIGH** arrival rate - **SET 1**

	SET 1			
R	**LL**	90thPC	**2nd Best**	90thPC
	2 Shards			
2	**537,184**	1,023,619	616,092 (RR)	1,161,649
3	**163,671**	299,093	238,846 (QL)	435,607
4	**127,450**	255,331	206,368 (RR)	396,222
5	**2,164**	4,331	57,493 (QL)	104,604
6	**9,269**	22,735	70,653 (QL)	142,159
7	**778**	2,069	2094 (QL)	4,326
8	**1,231**	3,224	12,818 (QL)	28,878
9	**644**	1,687	1,254 (QL)	3,022
10	**866**	2694	2,675 (QL)	6,167
	5 Shards			
2	**24,603**	48,737	201,014 (RR)	372,250
3	**471**	1,111	13,631 (QL)	24,092
4	**575**	1,410	13,117 (QL)	26,744
5	**364**	1,082	863 (QL)	1,855
6	**473**	1,293	1561 (QL)	3,743
7	**360**	1,082	621 (RR)	1,450
8	**469**	1,293	1,038 (QL)	2,539
9	**360**	1,082	493 (RR)	1,259
10	**469**	1,293	875 (QL)	2,062
	10 Shards			
2	**416**	929	62,678 (QL)	117,476
3	**250**	671	765 (QL)	1,536
4	**302**	711	1,071 (QL)	2,496
5	**247**	665	431 (RR)	952
6	**301**	711	646 (QL)	1,594
7	**247**	665	340 (RR)	792
8	**301**	711	551 (QL)	1,146
9	**247**	665	290 (RR)	721
10	**301**	711	443 (RR)	1,053

Table 4.10: Average Completion Times - **HIGH** arrival rate - **SET 2**

	SET 2			
R	**LL**	90thPC	**2nd Best**	90thPC
	2 Shards			
2	**390,769**	710,141	465,891 (QL)	844,484
3	**163,671**	299,093	238,846 (QL)	435,607
4	**50,325**	93,172	125,480 (QL)	227,770
5	**2,164**	4,331	57,493 (QL)	104,604
6	**1,028**	2,545	14,842 (QL)	27,172
7	**778**	2,069	2,094 (QL)	4,326
8	**683**	1,808	1,502 (QL)	3,370
9	**644**	1,687	1,254 (QL)	3,022
10	**628**	1,673	1,162 (QL)	2,890
	5 Shards			
2	**1,184**	1,354	122,056 (QL)	26,945
3	**471**	1,111	13,631 (QL)	24,092
4	**381**	1,088	1254 (QL)	2,481
5	**364**	1,082	863 (QL)	1,855
6	**361**	1,082	711 (RR)	160
7	**360**	1,082	621 (RR)	1,450
8	**360**	1,082	545 (RR)	1,328
9	**360**	1,082	493 (RR)	1,259
10	**360**	1,082	460 (RR)	1,217
	10 Shards			
2	**281**	691	14,630 (QL/RR)	1,144/1,137
3	**250**	671	765 (QL)	1,536
4	**247**	665	530 (RR)	1,129
5	**247**	665	431 (RR)	952
6	**247**	665	373 (RR)	854
7	**247**	665	340 (RR)	792
8	**247**	665	311 (RR)	759
9	**247**	665	290 (RR)	721
10	**247**	665	276 (RR)	698

Chapter 5

Hybrid Query Scheduling

5.1 Introduction

Chapter 4 has shown that, by making use of query efficiency predictors, a broker can appropriately schedule queries to the replicas most likely to be ready first, based on the expected duration of queries currently queued on each replica for processing. Indeed, Least Loaded was shown to improve over Queue Length and Round Robin and to be extremely similar to an oracle that has a priori knowledge of the actual response time for each query. This way, LL reduces the time that a query must spend waiting in a queue until it can be processed, with resulting benefits in query throughput.

However, where the contention for replicas is low, previous chapter has suggested that scheduling using efficiency predictions is unnecessary. LL introduces an overhead in the calculation of the predicted time for each query that can exceed the time that the query would have spent in a replica's queue while waiting to be processed. This is more likely to happen for query volumes that are adequately handed by the number of available replicas. Indeed, for large numbers of shards and replicas or low query traffic, we have noted no benefit in applying LL for query scheduling. This means that prediction is not always the best option to schedule the queries.

For this reason, a hybrid scheduling method is proposed, that brings the advantages of scheduling using query efficiency prediction under high load, whilst retains simplicity and speed under low load. In doing so, it adapts to the current query volume being experienced by the search engine.

Therefore, in this chapter we perform an exhaustive analysis of query scheduling methods, by comparing their performance under not only different numbers of shards and replicas but also under different query traffic flows. Indeed, as the volume of query traffic varies throughout the day [88], it is important to attain efficient retrieval with the minimum of resources.

Experiments are conducted using a proven simulation framework [23], prepared with the actual and predicted response times for 500,000 queries. Indeed, compared to previous chapter, these new experiments are conducted using a much larger corpus (ClueWeb09), and with a larger query set, issued to a Web search engine over the course of an entire day.

The remainder of this chapter is structured as follows: in Section 5.2, we discuss how queries can be scheduled among replicated query servers and propose a hybrid scheduling method that can adapt to changing query traffic conditions; Section 5.3 and 5.4 present the experimental setup and results, respectively; concluding remarks follow in Section 5.5.

5.2 Proposal

We hypothesize that due to the varying nature of the query traffic, one scheduling method is not appropriate for all times. Hence, we propose a hybrid approach that changes the scheduling method applied based on the current loading of the system. For measuring the current loading of the system, we use the moving average waiting time experienced by queries. In particular, if the current waiting time of queries is greater than the average cost of making an efficiency prediction, then we hypothesize that there is benefit in using LL over the simpler RR or QL methods. For instance, when combining RR for low traffic with LL for high traffic (which we denote **RR/LL**), the scheduling decision is:

$$schedule_{RR/LL}(q) = \begin{cases} RR & \text{if } \overline{WT} < \overline{PT}; \\ LL & \text{otherwise.} \end{cases}$$

The same for Queue Length (**QL/LL**):

$$schedule_{QL/LL}(q) = \begin{cases} QL & \text{if } \overline{WT} < \overline{PT}; \\ LL & \text{otherwise.} \end{cases}$$

In practice, $\overline{PT}$, the average time to make a prediction, can be estimated using training data, while $\overline{WT}$, the average time that queries are waiting at query servers to be processed, is computed using the moving average of the maximum time that a query spends queued for any shard, calculated over the last m minutes. The use of the m minute moving average prevents changes to the scheduling method for small bursts of query traffic. Increasing m has the effect of "smoothing out" the impact of larger variations on the choice of scheduling method, but can delay the onset of LL when appropriate.

To investigate the appropriateness of our proposed hybrid scheduling method, we conduct experiments using a real daily query flow comprising varying query volume throughout the day. These experiments are conducted within the context of a simulation framework, such as the one used in Chapter 4, which permits the varying of different number of shards and replicas, without the need for repeated large-scale distributed experiments.

In the following section, we define the experimental setup of the real IR system for which the various parameters of the simulated IR system are determined.

5.3 Experimental Setup

The experiments in the following sections are conducted using the large TREC ClueWeb09 category B corpus, which consists of 50 million Web documents, and aims to represent the first tier index of a commercial Web search engine. We index ClueWeb09 using the Terrier IR platform [65][1], using different numbers of shards, namely 2 and 5. Documents are partitioned across shards as per their ordering in ClueWeb09, which approximates crawl order. For each index, we apply Porter's English stemmer and removing standard stopwords. We also build skip lists for the inverted indices [77], with a skip pointer every 1,000 postings.

[1] `http://terrier.org/`

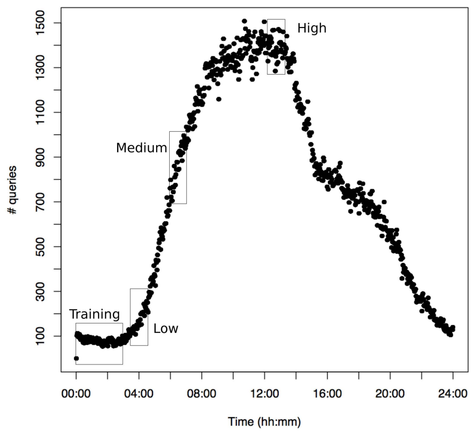


Figure 5.1: Query distribution in a 24 hours time span covering 1st May 2006.

Table 5.1: Subsets of queries selected for experimentation.

Subset	Frequency	Time interval
Low Volume	1.64 queries/sec	3:30am - 4:30am
Medium Volume	7.25 queries/sec	6:00am - 7:00am
High Volume	11.72 queries/sec	11:30am - 12:30pm

Best practices in efficiency experiments demand a large number of queries. In this work, we use a large sample of consecutive user queries from a publicly available real search engine log. In particular, we select roughly 500,000 consecutive queries from the MSN 2006 query log [34]. The selected queries exhibit all of the expected properties of a query log, such as frequently repeated 'head' queries and a tail of infrequent queries. Moreover, the query volume varies throughout the course of the (US west-coast) day - indeed Figure 5.1 shows the query arrivals in a day of activity (sampling rate 120 seconds).

We split these queries as follows: we use the first 5,000 queries (equating to the period 0:00am - 3:14am) as training for the efficiency predictors, while remaining queries are used for testing purposes. Moreover, to permit analyses of the scheduling accuracy under different query volumes, we select three subsets we indicate in Table 5.1. Each of these subsets are denoted in Figure 5.1.

We measure the response time for this setup for each index shard. Timing are made using a quad-core Intel Xeon 2.4GHz, with 8GB RAM, with inverted indices are stored on a 160GB SATA drive.

To compare the five scheduling methods, we use the same measures as in previous Chapter: average waiting time (AWT) and average completion time (ACT) for all the queries, in milliseconds (ms).

5.4 Results

In this section, we experiment to validate the hypothesis defined in Section 5.2 concerning hybrid scheduling. With this purpose, this section is structured into three subsections: Section 5.4.1 analyses the effect of the number of shard and replicas over the behaviour of the hybrid methods; Section 5.4.2 attests the influence of the incoming query traffic and Section 5.4.3 studies the importance of the moving average window value (m).

5.4.1 1st Factor: Architecture

In particular, Table 5.2 reports the Average Completion Time (ACT) and Average Waiting Time (AWT) of the three known scheduling methods – namely Round Robin (RR), Queue Length (QL), and Least Loaded (LL) (this includes the prediction costs) – for different configurations of shards and replicas. In addition, results for the proposed hybrid scheduling methods, Round Robin with Least Loaded (RR/LL) and Queue Length with Least Loaded (QL/LL), using a default moving window size of $m = 8$ minutes, are reported. We have not considered Random method into the experiments due to the low quality of the results experienced in previous chapter. The best scheduling method for each configuration is highlighted.

Firstly, we consider only the three non-hybrid scheduling methods. We observe that for settings with more contention for resources (i.e., small number of replicas and shards), RR and QL scheduling methods result in AWTs and ACTs somewhat worse than those obtained by LL, as well as the hybrid scheduling methods. When there is very little contention for query server resources (e.g., with 5 shards and 15 or more replicas), we obtain very low AWTs for RR and QL and broadly similar completion times. Nevertheless, LL presents slightly higher ACTs and AWTs due to the cost of making response time predictions.

Table 5.2: ACTs and AWTs for different shard and replica configurations, in ms. RR and QL times for 2 shards, 5 replicas are omitted, as this configuration cannot service the peak time query load. Prediction delay is included in the waiting time.

	ACT					AWT				
# Rep	RR	QL	LL	RR/LL	QL/LL	RR	QL	LL	RR/LL	QL/LL
	2 Shards									
5	-	-	**908**	**908**	**908**	-	-	**388**	**388**	**388**
10	683	650	**550**	555	554	162	130	**30**	34	34
15	564	559	**535**	542	541	44	39	**14**	21	21
20	537	536	**534**	**534**	**534**	16	15	14	14	**13**
	5 Shards									
5	716	636	**417**	**417**	**417**	346	303	**47**	**47**	**47**
10	424	421	**383**	384	392	54	51	**14**	**14**	22
15	380	**379**	383	379	**379**	10	10	14	**9**	**9**
20	**372**	**372**	537	**372**	**372**	3	**2**	14	3	**2**

Next, we consider the results for the hybrid methods (RR/LL and QL/LL), and observe that – as expected – the hybrid scheduling methods present an intermediate behavior, in the sense that they perform closer to LL in high contention settings, and they improve on LL's results in low contention settings. The behavior of the two hybrid methods, RR/LL and QL/LL, is similar across the various configurations of shards and replicas.

5.4.2 2nd Factor: Query volume

In order to analyse the effect of query volume across the different scheduling methods, Tables 5.3, 5.4 and 5.5 report the ACTs and the AWTs obtained in different scenarios during the low, medium and high query volume subsets respectively. Indeed, while for low query volumes all scheduling methods perform quite well (with the exception of LL, where the prediction cost of 13 ms is dominant), the benefits of LL and relative hybrid methods show up when the query traffic rises to medium and high volumes. In particular, the hybrid methods obtain better performance than LL when the prediction delay is significant (e.g., low contention). In this way, the hybrid scheduling methods are shown to be adapting to the current query volume.

More specifically, looking at table 5.3, it is clear to see the identical behaviour of RR, QL and the hybrid methods. For example, if we focus in configuration of 2 shards and 15 replicas, all the afore mentioned achieve an ACT of 472 ms, while LL exceeds this value with 485 ms. For 5 shards and 15 replicas, this behavior is kept – 349 ms of LL versus 336 ms achieved by the other approaches.

In case of medium query traffic, table 5.4 attests the behaviour of the hybrid methods: for a lower number of machines, hybrid methods imitates the performance of LL (i.e. 390 ms using 5 shards and 5 replicas, while the simplest methods expend more than 500 ms in answering the queries. Nevertheless, for a higher number of machines, hybrid methods shift to the simplest methods behaviour (i.e. with 5 shards and 20 replicas LL presents an ACT of 369 ms, meanwhile the rest of approaches reduce the ACT to 357 ms).

When the contention of the system increases (see table 5.5), the elastic behaviour of the hybrid methods continues, but in this case, RR/LL and QL/LL also imitate LL for some configurations with high number of machines as the one with 2 shards and 20 replicas.

To further aid analysis, this adaptive behavior of the QL/LL hybrid scheduling method is shown in Figures 5.2 and 5.3 (RR/LL produce results very similar to QL/LL, confirming earlier observations that QL insufficiently accounts for the varying response times of different queries when calculating the backlog of replicated query servers). In particular, in Figure 5.2 the QL/LL scheduling method clearly reproduces the scheduling method with the lowest AWT values, i.e. QL before 06:00 and after 16:00, and LL during peak daytime periods. In this scenario, the hybrid method achieves a total improvement in AWT of 68% over QL. On the other hand, in Figure 5.3, for high query load and small contention, this behavior is not evident, as both LL and QL have similar performance (less than 5ms difference in AWT, which is lower than the mean prediction delay of 13.52 ms). Hence, as there is little benefit for the hybrid method to switch to LL scheduling even under relatively high traffic load, it only does so for two very busy periods: mid-morning, from 10:40 to 11:10 and midday, from 11:40 to 12:50. In this case, the achieved total improvement in AWT is 7% with respect to QL.

Table 5.3: ACTs and AWTs for different for LOW query traffic, in different configurations. RR and QL times for 2 shards 5 replicas are omitted, as this configuration cannot service the peak time query load. Prediction delay is included in the waiting time.

ACT					
#Rep	RR	QL	LL	RR/LL	QL/LL
2 Shards					
5	-	-	487	**483**	**483**
10	**473**	**473**	485	**473**	**473**
15	**472**	**472**	485	**472**	**472**
20	**472**	**472**	485	**472**	**472**
5 Shards					
5	**340**	**340**	349	**340**	**340**
10	**336**	**336**	349	**336**	**336**
15	**336**	**336**	349	**336**	**336**
20	**336**	**336**	349	**336**	**336**
AWT					
2 Shards					
5	-	-	**2**	10	10
10	**0**	**0**	13	**0**	**0**
15	**0**	**0**	13	**0**	**0**
20	**0**	**0**	13	**0**	**0**
5 Shards					
5	4	4	13	4	4
10	0	0	13	0	0
15	**0**	**0**	13	0	0
20	**0**	**0**	13	**0**	**0**

Table 5.4: ACTs and AWTs for different for MEDIUM query traffic, in different configurations. RR and QL times for 2 shards 5 replicas are omitted, as this configuration cannot service the peak time query load. Prediction delay is included in the waiting time.

ACT					
#Rep	RR	QL	LL	RR/LL	QL/LL
2 Shards					
5	-	-	**684**	**684**	**684**
10	585	575	**523**	**523**	**523**
15	525	523	**518**	519	519
20	511	**510**	518	511	**510**
5 Shards					
5	550	533	**390**	**390**	**390**
10	380	379	**369**	372	372
15	**359**	**359**	369	359	**359**
20	**357**	**357**	369	**357**	**357**

AWT					
2 Shards					
5	-	-	**166**	179	179
10	81	70	**18**	18	18
15	20	19	**13**	15	14
20	**6**	**6**	13	**6**	**6**
5 Shards					
5	194	177	21	34	34
10	24	23	**13**	16	16
15	4	**3**	13	4	**3**
20	**1**	**1**	13	**1**	**1**

Table 5.5: ACTs and AWTs for different for HIGH query traffic, in different configurations. RR and QL times for 2 shards 5 replicas are omitted, as this configuration cannot service the peak time query load. Prediction delay is included in the waiting time.

ACT					
#Rep	RR	QL	LL	RR/LL	QL/LL
2 Shards					
5	-	-	**1214**	**1214**	**1214**
10	780	741	**565**	**565**	**565**
15	593	585	**485**	534	534
20	547	545	**533**	**533**	**533**
5 Shards					
5	959	877	**433**	**433**	**433**
10	467	462	**383**	**383**	**383**
15	389	388	**383**	385	385
20	374	**373**	383	374	**373**

AWT					
2 Shards					
5	-	-	**681**	695	695
10	280	221	**46**	**46**	**46**
15	74	66	**15**	**15**	**15**
20	28	26	**14**	**14**	**14**
5 Shards					
5	589	507	50	63	63
10	98	93	**14**	**14**	**14**
15	19	18	**14**	15	15
20	**4**	**4**	14	**4**	**4**

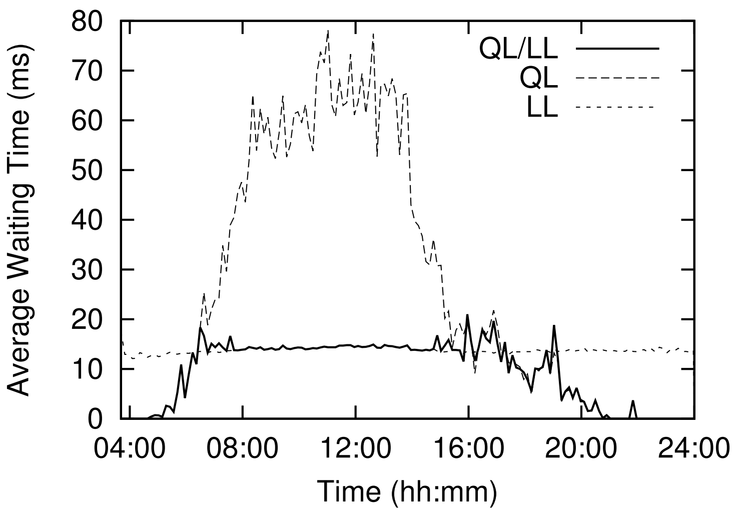


Figure 5.2: AWTs using 2 shards and 15 replicas. RR and RR/LL are omitted, as they produce results very similar to QL and QL/LL respectively.

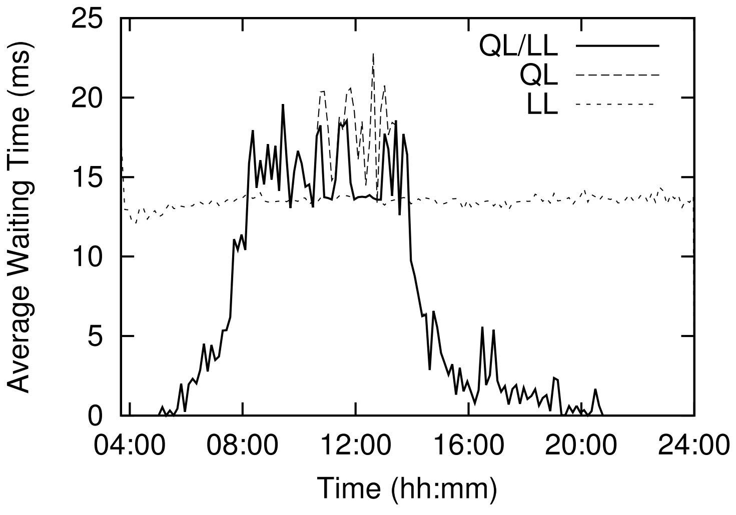


Figure 5.3: AWTs using 5 shards and 15 replicas. RR and RR/LL are omitted, as they produce results very similar to QL and QL/LL respectively.

5.4.3 3rd Factor: moving average window m

Finally, Table 5.6 includes the ACTs for the two hybrid methods under different scenarios, but according to a new parameter: the value (in minutes) of the width m of the moving average window used to compute the ACTs, presented in Section 5.2. We experimented using several values for this parameter: 1, 2, 4, 8 and 16 minutes.

For many configurations, m has no effect in the results. Under periods of medium contention (e.g. 15-20 replicas for 2 shards, and 10-15 replicas for 5 shards), results show benefit in ACTs by increasing m. Indeed, by increasing m, the scheduling method does not oscillate between scheduling methods for short bursts of high query volume.

Table 5.6: ACT (in milliseconds) using different sizes of moving average window (m) for both hybrid methods and combination of shards and replicas.

	RR/LL					QL/LL				
m	1	2	4	8	16	1	2	4	8	16
#Rep	2 Shards									
5	**908**	**908**	**908**	**908**	**908**	**908**	**908**	**908**	**908**	**908**
10	**550**	**550**	**550**	**550**	**550**	**550**	**550**	**550**	**550**	**550**
15	537	535	**533**	**533**	**533**	536	534	533	533	**530**
20	533	532	**531**	**531**	**531**	532	531	531	**530**	**530**
#Rep	5 Shards									
5	**417**	**417**	**417**	**417**	**417**	**417**	**417**	**417**	**417**	**417**
10	393	389	386	384	**383**	392	388	386	**383**	**383**
15	**379**	**379**	**379**	**379**	**379**	379	379	379	379	**378**
20	**372**	**372**	**372**	**372**	**372**	**372**	**372**	**372**	**372**	**372**

5.5 Conclusions

Query scheduling addresses the need for a distributed IR system to select the replicated query server for a new query that will be available to process it. Previous chapter has shown the advantages of query scheduling while making use of query efficiency predictors to accurately estimate the workload of a replicated query server. However, the use of query efficiency predictors introduces an overhead that can hinder on scheduling accuracy during periods of low contention for query server resources (e.g. low query traffic times, such as at night). In this chapter, we addressed this limitation by proposing a new hybrid scheduling method, which adopts the behavior of prediction query scheduling for high query traffic, but resorts to a lightweight scheduling method – such as Round Robin – when there is less contention. In this way, the scheduling can adapt as the query volume varies throughout a typical day whilst minimizing the waiting times experienced by queries.

Our experiments compare the proposed hybrid query methods with lightweight scheduling methods. This is performed using a realistic Web search setting, involving ClueWeb09 and 500,000 queries submitted to a real Web search engine over the course of an entire day. Our results show that hybrid scheduling can reduce the waiting time experienced by search engines across varying traffic conditions, by reducing delays when there is low contention (up to 7% AWT reduction), and by choosing a more complex but highly efficient scheduler for the busiest periods of the day (up to 68% AWT reduction).

Once we have achieved a great improvement regarding the latency of a search engine, next chapter aims to address IR system resource provisioning [89], such that the desired efficiency can be attained at all periods while minimizing the number of powered on query servers, with potential power consumption savings.

Chapter 6

Power/Latency Trade-off Model

6.1 Introduction

Previous chapters have shown that user query volume typically received by a Web search engine varies through the course of the day [88]. In order to guarantee that each query is processed with sub-second response times, the computing/communication infrastructure has to support worst-case query volume, which typically reaches its maximum during the day time. Hence, the typical approach taken by Web search engines is to deploy a distributed search architecture relying on a very large data centre to deal with the worst-case query volumes [11]. The main goal of this high-level approach is to maximise the query throughput of a search engine, providing users with effective query results in a timely manner. Supporting worst-case query volumes in large data centres has the obvious drawback that the power consumption/electricity costs are not taken into account, resulting in a potential waste of power and money when the query volume is low.

We argue that a trade-off can be enforced between the machines devoted to processing queries and query processing deadlines, and that this trade-off can be adapted during the operational cycle of a Web search engine, in order to minimise the number of machines processing the queries while ensuring acceptable latencies. Moreover, we contend that this trade-off can be adapted dynamically to changing query volumes in very short times.

The machines that are not used when the query volume is low could be used to execute different tasks, involved in the generation and management of search results, such as batch data processing, ads/recommendations generation, indexing, and so on.

To the best of our knowledge, no previous work has presented an intra-data centre model that turns the servers on or off depending on the incoming query traffic needs. Indeed, our framework can examine the historical and current query traffic patterns to predict the number of query server replicas now needed, and obtain power savings within a single data centre by eliminating query servers that are not currently needed. Moreover, we provide thorough experiments using 1 million queries submitted to a real Web search engine over the course of 2 days, to demonstrate the power savings that can be obtained without marked impact on the efficiency of the search engine. Our results show that our proposed self-adapting model can achieve an energy saving of 33% while only degrading mean query completion time by 10ms compared to a baseline that provisions replicas based on a previous day's traffic.

The remainder of this chapter is structured as follows: Section 6.2 presents the lack of previous works concerning power consumption of IR systems, and this constitutes the main motivation of our work. In Section 6.3 we concretely state our proposal and research questions. In Section 6.4, we introduce our model, by describing the dynamic system and the general cost function, while Section 6.5 presents the deterministic approach that considers the previous and also the current state of the system for predicting query traffic. Sections 6.6 and 6.7 propose power and latency cost functions, respectively. Section 6.8 details the baselines and experimental setup. Our experimental results are reported in Section 6.9. Section 6.10 presents the generalization of the model for distributed and replicated architectures, as well as different scheduling methods. Finally, concluding remarks are summarised in Section 6.11.

6.2 Background

Many of the works mentioned in Chapter 2 (Section 2.5) are focused on achieving power savings in data centre for general Internet services (e.g. [59, 74]). However, few works have addressed this problem within search engines, where user queries equate to (short-lived) jobs. Within the roadmap proposed by Chowdhury [32], which introduced the term *Green IR*, our work is clearly focused on the intra-data centre efficiency of the search engine. In contrast, the work of [53] addressed power efficiency at the inter-data centre level, by distributing query volume between geographically distant data centres based on workload and electricity prices. Recently, Sazoglu et al. [86] propose a novel metric for result caching that considers the financial cost of a cache miss.

Nevertheless, to the best of our knowledge, no previous work has presented an intra-search engine model for managing the powering of the servers depending on the incoming query traffic needs.

6.3 Proposal

To attain the problem shown in previous section, we start our research by stating the following hypothesis: *It is possible to* dynamically adapt *the behaviour of the search engine – according to the variations of the query load – while providing acceptable query latencies and minimising the number of machines used to process the queries*.

We propose a mathematical model of a replicated search engine with a query broker and many independent query processors, each managing a replica of the index. This model permits the number of query processors to be dynamically changed according to the query arrivals and proposed latency and power consumption cost functions. By estimating the arrival times and processing requirements of future queries, we derive self-adapting mechanisms for the search engine model that can reduce power consumption without negatively impacting efficiency, by means of dynamic optimisation schemes [13].

6.4 Mathematical Model

6.4.1 Architecture

As we could see in previous sections, increasing the parallelism of a search engine through distributed architectures offers a route to increased per-request efficiency without loss of effectiveness. In such *document-partitioned* architectures, a query server stores the index shard for a subset of the documents in the corpus. Indeed,

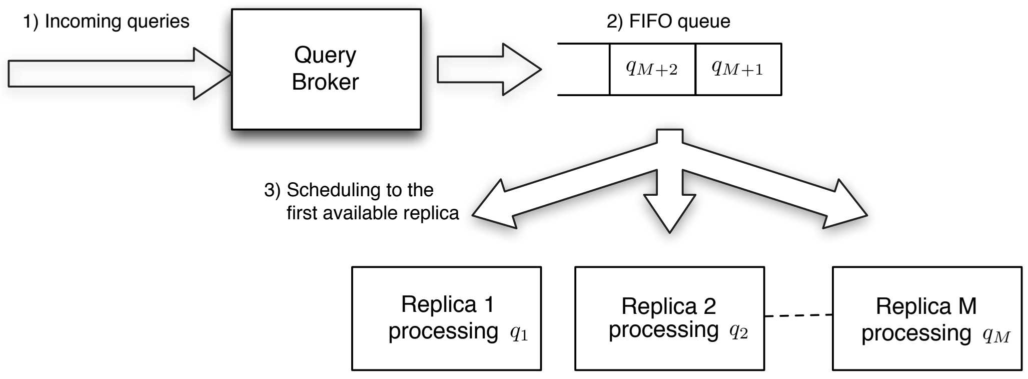

1) Incoming queries
Query Broker
2) FIFO queue
q_{M+2}
q_{M+1}
3) Scheduling to the first available replica
Replica 1 processing q_1
Replica 2 processing q_2
Replica M processing q_M

Figure 6.1: Our reference architecture.

with multiple *replicas* of the same shard, more queries can be processed in parallel on identical shard copies, thus reducing the waiting time of incoming queries, as well as providing fault tolerance properties. For simplicity, in this section we focus on a single broker, single-shard environment with multiple replicas on different query server machines. Indeed, as the replicated query servers allocated to a single shard represent independent partitions of the search engine's index, the techniques proposed in this section could easily be applied for multiple shard environments, by independent application on each shard individually (as it will be attested in section 6.10). Nevertheless, for simplicity, we are formulating the model assuming a one-shard architecture.

Our reference architecture assumes a single-shard search engine implemented by a query broker and M independent replicas that manage a copy of the index (see Figure 6.1). User queries are received by the broker and are queued up in a buffer. The queries stored in the buffer retain their positions until they are completed. While complex queue re-ordering strategies can benefit overall response times [43, 67], for simplicity, in our architecture the query processing nodes serve from the front of the queue: at any given time if the search engine has less than M queries, some processing nodes are idle, while if the search engine has more than M queries, some queries are queued in the buffer.

6.4.2 Dynamic Optimisation Model

We consider a replicated search engine processing user queries, with a single *query broker* and M independent identical *query processors* each serving a replica of the index (as explained above with reference to Figure 6.1). As in previous chapters, each query submitted to the search engine experiences a completion time defined as the sum of its waiting time (the time that the query has spent enqueued), its processing time (the time spent processing the query by a query processor) and any network delays between the broker and the corresponding replica. Assuming identical query processors, the processing time of a query is independent of the node actually processing the query: the same query will be processed in the same amount of time on each node.

We consider a daily-based operational cycle of the search engine, in that we analyse the behaviour of the search engine during a single day. Periodically during the day, we *observe* the state of the search engine and we *decide* how to change it, with the objective of minimising a certain cost, i.e., an undesirable behaviour. In our scenario, this behaviour is represented by the unnecessary usage of machines that are not necessary to service the query load with acceptable timeliness. In doing so, we must take into account that the outcome of each decision cannot be fully predicted, due to some random unknown parameters – such as the number of queries that will be received. Moreover, each decision cannot be taken in isolation, since we want to balance lowering the present cost with potentially higher future costs – for instance, turning off currently unused machines that might be needed shortly. To achieve these aims, we model the search engine as an optimal decision problem of a discrete dynamic system over a finite number of stages [13]. Computing systems have been previously modelled as dynamic systems in order to leverage automatic control theory to address the dynamics of resource management [49], such as email server [79] and web servers [36]. However, this work represents the first instantiation of a dynamic decision problem within search engine power/latency modelling.

In the remainder of this section, we provide a short introduction to the general dynamic decision problem [13] (Section 6.4.3), a dynamic model of a replicated search engine with multiple query processors (Section 6.4.4), a discussion on the

Table 6.1: Notation used within our model.

Symbol	Explanation
N	number of time slots (per day)
T_s	length of a time slot (in secs)
M	number of available machines
x_k	queued queries at the beginning of time slot k
u_k	processing nodes during time slot k
w_k	incoming queries during time slot k
y_k	processed queries at the end of time slot k
$\bar{w}_k$	estimated incoming queries during time slot k
v_k	mean query processing time during time slot k
$\bar{v}_k$	estimated mean average query processing time during time slot k
$f_k(\cdot)$	generic state update function
$g_k(\cdot)$	generic cost function
$P_k(\cdot)$	power cost function
$L_k(\cdot)$	latency cost function
$h_k(\cdot)$	query processing function

cost function for our dynamic model (Section 6.4.5), and a summary of the resulting decision problem (Section 6.4.6).

6.4.3 General Dynamic Decision Problem

A dynamic decision problem model must be composed by: (1) an underlying *discrete-time dynamic system* and (2) a *cost function that is additive over time* [13]. In the following, we introduce the notation necessary to describe a general decision problem model, which we later instantiate for our proposed search engine model. All notation used in our instantiation for a search engine problem model is summarised in Table 6.1. Firstly, we assume that time is slotted and indexed by $k = 0, 1, 2, \ldots, N$. Time slots are sampled every T_s seconds. The dynamic system has the form:

$$x_{k+1} = f_k(x_k, u_k, w_k) \qquad k = 0, 1, \ldots, N-1 \tag{6.1}$$

where k indexes discrete time, x_k represents the state of the system that is relevant for its future operation, u_k is the decision variable to be selected at time k and w_k is a random parameter. In general, we deal with a *finite time horizon*, i.e., we observe and optimise the system during a fixed number of time slots, indexed from 0 to N – for instance, over a 24 hour period. The random parameter (or noise, or disturbance, or exogenous input) w_k is characterised by a probability distribution that may depend explicitly on x_k and u_k but not on the values of prior disturbances $w_0, \ldots, w_{k-1}$. Given an initial state x_0 and a sequence of decisions $u_0, \ldots, u_{N-1}$, the states x_k and the disturbances w_k are random variables with distributions defined through Equation (6.1).

The cost function defines the expected cost of the decision at time k, and is additive in the sense that the cost incurred at k, denoted by $g_k(x_k, u_k, w_k)$, accumulates over time. Note that g_k is a random variable, since it depends on x_k and w_k. Hence the expected total cost $J(x_0)$ is:

$$J(x_0) = E\left[g_N(x_N) + \sum_{k=0}^{N-1} g_k\big(x_k, u_k, w_k\big)\right] \tag{6.2}$$

where the expectation is taken over the random variables x_k and w_k and $g_N(x_N)$ is a terminal cost incurred at the end of the process, depending on the final state. Hence, an *optimal decision sequence* $u_0^*, \ldots, u_{N-1}^*$ is the decision sequence that minimises the cost $J(x_0)$.

6.4.4 Search Engine Dynamic Model

Following the above formulation of a general dynamic decision problem, we now instantiate a dynamic decision model for a search engine. In each time slot k, depending on the number of pending queries in the buffer, the search engine allocates a number of processing nodes among the M available. Increasing the number of processing nodes decreases the overall waiting time for the queries in the buffer but increases the service cost and power consumption associated with the system.

At the beginning of each time slot, a decision must be taken regarding the number of processing nodes to be used in that slot. Decisions cannot be viewed in isolation since we want to balance two conflicting goals: minimising the power consumption of the search engine, and maximising the search engine's efficiency. We denote by:

- x_k: the number of queries waiting to be processed (i.e., currently enqueued) at the start of the kth time slot.
- u_k: the number of *active* processing nodes at the start of the kth time slot.

- y_k: the number of queries processed by the search engine during the kth time slot (with a given probability distribution).
- w_k: the number of queries arriving to the search engine during the kth time slot (with a given probability distribution).

We assume that any incoming query is queued, and that if there are waiting queries, any processing node that finishes to process a query immediately starts processing a waiting query. Thus, the number of queries waiting to be processed evolves according to the following discrete-time equation:

$$x_{k+1} = x_k - y_k + w_k \tag{6.3}$$

The dynamic equation (6.3) does not explicitly depend on the number of active processing nodes u_k. However, the number of queries processed by the search engine during the kth time slot y_k depends explicitly on u_k, according to the following general equation:

$$y_k = h_k(x_k, u_k, v_k) \tag{6.4}$$

This models the fact that, in a given time slot, the number of processed queries depends on three quantities: the number of queries waiting to be processed x_k, the number of processing nodes u_k, and a random component v_k modelling the mean service time of the queries. To summarise, the complete model for the search engine under examination is:

$$x_{k+1} = x_k - h_k(x_k, u_k, v_k) + w_k \tag{6.5}$$

where we leave to later in Section 6.5 the specification of the query processing function $h_k(x_k, u_k, v_k)$, which defines how many queries of x_k can be processed by u_k machines with a given service time pattern of v_k.

6.4.5 Search Engine Cost Function

A public software service such as a search engine has two main stakeholders: the service provider and the service user. In general, the service provider aims to maximise the revenue from the service, increasing the income and reducing its operating expenditure. One of the main costs of a running service is the expenditure on power required to run the machines hosting the service. From the point of view of the user, a search service is desired to be efficient (timely), i.e., the latency between the submission of a query and the display of the first search results should be minimised.

Clearly, the operating cost of a search infrastructure depends on the number of machines operating the service, and the final revenue depends on the number of satisfied users, i.e. how many users receive their search results with acceptable latency. Indeed, search engines users have less tolerance for slower search engines [87], and may abandon their search request [56]. Such abandonment can lead to the loss of users to other search services, leading to a loss of potential revenue from these users.

Given that the number of queries submitted to a search engine varies during the day [88], the number of processing nodes can be varied according to the demand. At the start of each time slot, a decision regarding the number of processing nodes to be used must be taken. Decisions need to balance two conflicting goals: minimise the search engine power consumption and maximise the search engine efficiency (by reducing latency). Hence, we have two types of costs that should both be considered:

1. Power cost $P_k(x_k, u_k, v_k, w_k)$, increasing in the number of processing nodes used;

2. Latency cost $L_k(x_k, u_k, v_k, w_k)$, increasing in the number of queries waiting to be processed and decreasing in the number of processing nodes used.

As the latency costs increase, the model aims to minimise the overall cost by emptying the query queue faster. On the other hand, as the power costs increase, the system will attempt to trade higher latencies for lower power. In order to model this power/latency trade-off, we propose cost combinations of the following type:

$$g_k(\cdot) = \lambda P_k(\cdot) + (1 - \lambda) L_k(\cdot) \tag{6.6}$$

for various values of $\lambda \in [0, 1)$. For $\lambda = 0$, the cost function represented by Equation (6.6) ignores any power cost, and leads to the maximum number of available processing nodes being used in every time slot, as this achieves the minimum possible queueing delay. If $\lambda = 1$ is allowed, the cost function would ignore any latency cost, leading to the limit case of no processing nodes being used for processing, thereby maximising power savings but leading to infinite waiting times. Varying λ in $[0, 1)$, we can achieve any average query latency from infinite to the minimum possible traded off against the corresponding power consumption of the search system. We note that both cost functions assume values in the same range. Without loss of generality, later in Sections 6.6 and 6.7, we devise particular cost functions ranging in the [0,1] interval, where 0 means no cost and 1 means maximum cost.

6.4.6 Latency/Power Decision Problem

We now formulate the general dynamic decision problem by adapting Equations (6.1) & (6.2) to our search engine and cost models:

$$\begin{aligned} \underset{u_k}{\text{minimise}} \quad & E\Bigg[\sum_{k=0}^{N-1} \lambda P_k(x_k, u_k, v_k, w_k) + \\ & (1-\lambda) L_k(x_k, u_k, v_k, w_k) \Bigg] \\ \text{subject to} \quad & x_{k+1} = x_k - h_k(x_k, u_k, v_k) + w_k, \\ & k = 0, 1, \ldots, N-1 \end{aligned} \tag{6.7}$$

As described above, λ is an exploratory parameter that must be fixed at the beginning, while w_k and v_k are random variables describing the number of incoming and processed queries at the kth time slot respectively. The state variables x_k are computed through N instances of Equation (6.5), depending on these two random variables and the query processing functions $h_k(\cdot)$. Given these dependencies, the state variables are random variables as well, hence the expectation in the cost function must be computed over these three sets of random variables, in a stochastic manner. The query processing functions $h_k(\cdot)$ will be defined in the next section, where we describe some approximations that allow the stochastic decision problem (6.7) to be sub-optimally solved in a deterministic manner.

6.5 Deterministic Approximation

Problems like (6.7) cannot typically be solved analytically, and their solution algorithms are computationally very intensive [13]. For these reasons, these problems, where the exact value of all variables are unknown deterministically, are solved suboptimally in practice. In order to deal with the stochastic decision problem (6.7), we propose to solve a suboptimal scheme that consists of computing, at each stage, a decision that would be optimal if the uncertain quantities were fixed at some typical values. In doing so, we replace the stochastic nature of the decision problem with a simpler *deterministic* version at each stage and then we solve the deterministic problem. Within this section we discuss:

1. how to estimate the 'typical' value of the random variables, which we denote by $\bar{w}_k$ and $\bar{v}_k$, representing the estimated number of queries arriving during the kth time slot and the estimated mean service time during the kth time slot (Section 6.5.1);

2. how to derive a deterministic approximation for problem (6.7) with query processing functions $h_k(\cdot)$ depending on our estimations (Section 6.5.2);

3. how to solve the deterministic problem using dynamic programming, if we know all the estimates of the random variables for the whole day, and how to solve the deterministic problem with simple subsequent steps, if we know the estimates of the random variables for the next time slot only (Section 6.5.3).

6.5.1 Random variables estimation

In order to compute the estimated values of the random variables w_k and v_k, we adopt the following estimation schemes, based on historical data.

For the service times of queries, the typical assumption is that these are independent and identically distributed random variables [19]. However, we assume that the service times exhibit a *seasonal* trend among days, hence the mean service time of the queries in the k-th time slot is equal to the mean service time of queries in the same time slot of a previous day, i.e., $\bar{v}_k = v_{k-JN}$, where J defines the number of previous days, and N is the total number of time slots in a day. For instance, $J = 1$ means that $\bar{v}_k$ is estimated using data from the previous day, while $J = 7$ means that $\bar{v}_k$ is estimated using data from the same day in the previous week.

For the number of queries arriving during the kth time slot, we will assume two different estimation schemes:

- In the *seasonal estimator*, we estimate the number of incoming queries with the actual number of incoming queries in the same time slot of a previous day, i.e.:

$$\bar{w}_k = w_{k-JN}$$

- In the *seasonal estimator with drift*, the previous day value is adjusted with the current trend of arrivals [82] experienced in the last two time slots, such that:

$$\bar{w}_k = w_{k-JN} + (w_{k-1} - w_{k-2})$$

While the seasonal estimator is a viable solution for days exhibiting the same query submission and execution patterns (e.g., two subsequent weekdays or the same day in two subsequent weeks), the seasonal estimator with drift takes into account potential and unpredictable changes in query volume patterns, such as reduced query volumes during holidays or increased query volumes during major events (e.g., disasters, breaking news, or sport events). However, for this estimator, the value of $\bar{w}_k$ is known for only the next time slot ahead.

6.5.2 Deterministic Problem Formulation

Given the mean query service time v_k, it is straightforward to model the processing of queries in our search engine model. If a single node can process T_s/v_k queries in the T_s seconds duration of the kth time slot, then u_k identical processing nodes can serve $u_k \cdot T_s/v_k$ queries during the same slot, resulting in the following expression for h_k:

$$h_k(\cdot) = u_k \cdot T_s/v_k \tag{6.8}$$

Since we use the estimated mean service time $\bar{v}_k$ instead of the actual mean service time v_k, the approximation of the stochastic decision problem (6.7) with the

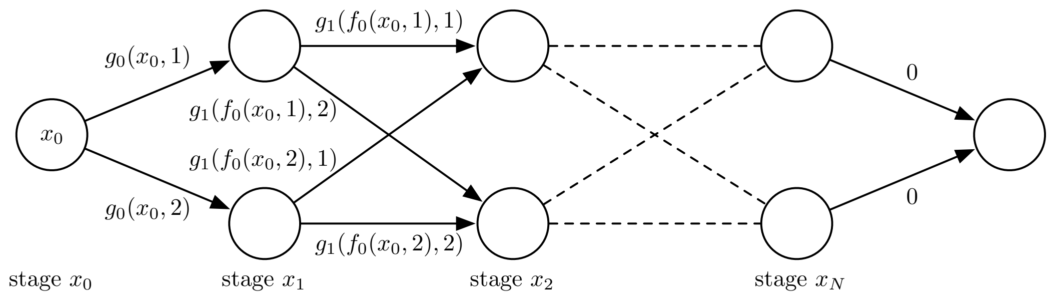


Figure 6.2: Transition graph for a deterministic problem with 2 machines.

random variables' estimated values leads to the following deterministic problem formulation:

$$
\begin{aligned}
\underset{u_k}{\text{minimise}} \quad & \sum_{k=0}^{N-1} \lambda P_k(x_k, u_k, \bar{v}_k, \bar{w}_k) + \\
& (1-\lambda) L_k(x_k, u_k, \bar{v}_k, \bar{w}_k) \\
\text{subject to} \quad & x_{k+1} = \max\{0, x_k - u_k \cdot T_s/\bar{v}_k + \bar{w}_k\} \\
& k = 0, 1, \ldots, N-1
\end{aligned}
\tag{6.9}
$$

where the $\max\{\cdot\}$ function avoids to process more queries than the number of queries available to process. In contrast to problem (6.7), problem (6.9) is deterministic as there is no longer any probability distributions associated with v_k and w_k (and hence x_k). Such deterministic formulation can be tractably solved [13], as described next.

6.5.3 Deterministic Problem Solutions

Consider the deterministic problem formulation defined in Equation (6.9) where each state x_k can assume a finite set of values. Then, at any state x_k, a decision u_k can be associated with a transition from state x_k to state $f_k(x_k, u_k) = x_k - u_k \cdot T_s/\bar{v}_k + \bar{w}_k$ at a cost $g_k(x_k, u_k) = \lambda P_k(x_k, u_k, \bar{v}_k, \bar{w}_k) + (1-\lambda) L_k(x_k, u_k, \bar{v}_k, \bar{w}_k)$. As illustrated in Figure 6.2, the deterministic problem can be equivalently represented by a graph, where the arcs correspond to transitions between states at successive stages and each arc has an associated cost corresponding to $g_k(\cdot)$. Decision sequences correspond to paths across the graph, originating at the initial state (node at stage 0, where $x_0 = 0$), and terminating at a final node linked to all terminal states (nodes at stage $N-1$) with no associated transition cost or, alternatively, with a cost proportional to the number of remaining unprocessed queries. If we view the cost of an arc as its length, we see that the deterministic problem of Equation (6.9) is equivalent to finding a minimum length path from the initial node (at stage x_0) to the artificial terminal node with no transition costs of the graph.

If the service times and the number of arriving queries are estimated using only historical data (e.g. based on a previous day, the seasonal estimator), then the transition costs of the whole graph for the current day can be computed a priori. Hence, it is possible to use dynamic programming to solve the general shortest path problem [13], but also algorithms specifically designed for the shortest path problem solution, such as the Dijkstra algorithm [37]. We denote this solution algorithm as LONGTERM.

On the other hand, if we estimate the number of arriving queries with the seasonal drift approach, the estimates of the number of queries w_k depend not just on the historical data, but also on the current load being experienced by the search engine. Hence, it is not possible to compute all the transition costs in the graph at the start of the day [13]. Instead, at each step we truncate the *estimation horizon* (i.e. how ahead the costs are calculated) to the next step only and resort to a *one step limited lookahead* strategy, where, at each stage, we select the next stage reachable with minimum cost from current stage[1]. We will denote this solution algorithm with SHORTTERM. Moreover, as discussed in Section 6.5.1, we expect SHORTTERM to improve LONGTERM, as it consider the query volume being currently experienced by the search engine in addition to the volume experienced on a previous day.

6.6 Power Cost Function

The power cost function $P_k(x_k, u_k, \bar{v}_k, \bar{w}_k)$ represents the electric power consumption of the whole search engine and it is directly proportional to the energy costs of operating the search engine. Firstly, we discuss the power usage of a single processing node. We distinguish between three states that a node can be in:

1. ON. The node is fully operational and busy processing a query. The node consumes power at a rate of P_{on}.

[1]We leave an examination of strategies with larger lookaheads to future work.

2. STANDBY. The node is available, but is currently sleeping. The node consumes power at a rate of P_{standby}.

3. OFF. The node is off, and it consumes no power.

Switching a node between two states is associated with a *switching cost*. The switching cost typically consists of two components: a time component and a power component. The time component depends on node characteristics and the search engine implementation, while the power component depends on the power consumed by the node during setup time (typically coinciding with P_{on}). While the switching time on $\leftrightarrow$ standby is almost instantaneous, the time required to switch between on and off and vice-versa is not negligible: for most data centres [45], this switching time can reach 200 seconds. This setup time can negatively impact on the latency of the queries to be processed by the node, and must be avoided. As the modelling of switching times is not considered in our cost functions, we limit our machine operational states to on and standby.

Given these costs, we assume that a fully operational node is consuming P_{on} Watts per T_s seconds, while a standby node is not turned off but consuming P_{standby} Watts per T_s seconds. So, at a given time slot k, the total energy consumed by a search engine with u_k active processing nodes out of a possible M is:

$$P_{\text{on}}T_s u_k + P_{\text{standby}}T_s(M - u_k)$$

Please note that this power consumption is an upper bound approximation of the actual power costs, because we are implicitly assuming that all active node will always be processing queries. By normalising this quantity by the maximum consumable power for M machines, we obtain the following expression for the power cost function $P_k(\cdot)$:

$$P_k(\cdot) = P(u_k) = \frac{1}{MP_{\text{on}}}\left[P_{\text{on}}u_k + P_{\text{standby}}(M - u_k)\right] \tag{6.10}$$

Note that the power cost function does not depend on k, x_k, $\bar{v}_k$ or $\bar{w}_k$, and that it varies between $P_{\text{standby}}/P_{\text{on}}$ and 1.

6.7 Latency Cost Function

At this point, we must define a function to represent the latency cost in order to make it comparable with the power consumption function. While searching for a representation of the time that a query has to spend waiting in a queue, it is naturally to think about queueing theory, as it provide us the value of two interesting variables: the average time that a query must spend in the system and also the estimated number of incoming queries. On the other hand, a more practical, less formal, but easier to estimate approach is also considered.

The remainder of this section is structured as follows: section 6.7.1 presents basic concepts on Queueing Theory and check its suitability into our scenario and section 6.7.2 builds a latency function using a deterministic approach based on historical/current data. Later in Results Section (6.9) we prove the suitability of these approaches.

6.7.1 Queueing Theory approach

In general, a *queue* can be defined as a waiting line (like customers waiting at a bank office) [33]. *Queueing Theory* deals with the analysis of waiting lines where customers wait to receive a service [29], [20]. More generally, Queueing Theory is concerned with the mathematical modeling and analysis of systems that provide service to random demands. A queueing model is an abstract description of such a system. Typically, a queueing model represents the following aspects [33]:

- The system's physical configuration, by specifying the number and arrangement of the *servers*, which provide service to the *customers*.
- The stochastic (that is, probabilistic or statistical) nature of the demands, by specifying the variability regarding the *arrival and services processes*.

Queueing theory is considered to be one of the standard methodologies (together with linear programming, simulation, etc.) of operations research and management science and is standard fare in academic programs in industrial engineering, telecommunications or computer science. There is huge research material on queueing theory, and it continues to be published at an increasing rate. But, despite its apparent simplicity, complexity and rigour are inborn characteristics.

A queueing model can be mainly characterized by the following parameters:

- Arrival rate (λ): mean number of arrivals per time unit. Interarrival rate: $\frac{1}{\lambda}$.
- Service rate (μ): mean number of clients that are served per time unit.
- Service capacity (s): number of servers helping the customers.
- Service discipline: First Come First Served (FCFS), Random, Last Come First Served (LCFS), etc.

Kendall [54] introduced a shorthand notation to characterize a range of these queueing models. Its simplest form consists in a three-part code $a/b/c$. The first letter specifies the interarrival rate distribution and the second one the service rate distribution. For example, for a general distribution the letter G is used, M for the exponential distribution and D for deterministic times. The third and last letter specifies the number of servers. Some examples are $M/M/1$, $M/G/1$, $G/M/1$ and $M/D/1$. We are using the model $M/M/s$ (on section 6.7.1.1 we test the availability of this model for solving our problem).

The universal notation of queueing theory also includes the following parameters:

- Service time:
$$\rho = \frac{\lambda}{s \cdot \mu} \tag{6.11}$$
If $\rho < 1$ the system is said to be stationary and the model is able to calculate a solution based on the following formulas.
- Probability of n clients to be in the system (in an stationary state) (P_n).

$$p_0 = \frac{1}{\sum_{n=0}^{s-1} \frac{(\frac{\lambda}{\mu})^n}{n!} + \frac{(\frac{\lambda}{\mu})^s}{s!(1-\rho)}} \tag{6.12}$$

$$p_n = \frac{(\frac{\lambda}{\mu})^n \cdot p_0}{n!}, 0 \leq n \leq s \tag{6.13}$$

$$p_n = \frac{(\frac{\lambda}{\mu})^n \cdot p_0}{s! s^{n-s}}, n > s \tag{6.14}$$

- Estimated number of clients in the queue (L_q).

$$L_q = \frac{(\frac{\lambda}{\mu})^s \cdot p_0 \cdot \rho}{s! \cdot (1-\rho)^2} \tag{6.15}$$

- Mean waiting time in the system (W).

$$W = W_q + \frac{1}{\mu} \tag{6.16}$$

- Mean waiting time in the queue (W_q).

$$W_q = \frac{L_q}{\lambda} \tag{6.17}$$

- Estimated number of clients in the system (L).

$$L = \lambda \cdot W = L_q + \frac{\lambda}{\mu} \tag{6.18}$$

6.7.1.1 Study of the viability of Queueing Theory to the current scenario

Before applying Queueing Theory to our scenario, we must previously test if our scenario satisfies the conditions to apply a $M/M/s$ model: we must check that the query interarrival times and also the query servers processing times follow an exponential distribution.

Regarding the interarrival times, our scenario differs from the one that uses the whole daily query times. We are applying one different $M/M/s$ model on each time slot (in this study, each 15 minutes), so we must test if the interarrival times belonging to a slot follows an exponential distribution. For this purpose, we have used SPSS software[2] to check the distributions of the 96 slots. If the number of queries per second follows a Poisson distribution, we can assume that the interarrival times follow a exponential distribution. Table 6.2 shows the parameters and results of the Hypothesis test. Based on these results we assume that the interarrival times of each slot follow an exponential distribution, so we can apply the $M/M/s$ model.

[2]`http://www-01.ibm.com/software/analytics/spss/`

Table 6.2: Results of the statistical analysis for the distribution of the number of queries per slot

Test	Kolmogorov-Smirnov
α (Significance Level)	0.05
#Contrasts (slots)	96
H_0 H_1	#Queries per slot fits a Poisson distribution #Queries per slot does not fit a Poisson distribution
#Contrasts that accept H_0	92

After this statistical analysis we assume that QT can be applied to our scenario. The basic parameters λ and μ are estimated as follows:

$$\lambda = \bar{w}_k \tag{6.19}$$

$$\mu = \frac{T_s}{\bar{v}_k} \tag{6.20}$$

In Eq. 6.20 T_s represents the length of the slot.

$M/M/s$ model gives us a formula to calculate the mean waiting time in the system (Eq. 6.16) based on the values of λ and μ. The latency function represents the time that the system will spend in solving all the queries within a slot. This way, we calculate the latency following Equation 6.21. Remember that s represents the number of replicas. This equation includes the negative exponential normalization.

$$L_k = 1 - e^{-(W \cdot T_s \cdot \bar{w}_k / s)} \tag{6.21}$$

6.7.2 Deterministic approach

The latency cost function $L_k(x_k, u_k, \bar{v}_k, \bar{w}_k)$ represents the cost incurred when the time required to process queries increases. In order to provide a simple analytic expression for this cost, consider the following situation. At the beginning of time slot k, we have x_k queued queries, waiting to be processed by u_k nodes with an average service time per node of $\bar{v}_k$ seconds. During the k-th time slot, we receive $\bar{w}_k$ new queries to process. We want to compute the average latency of $x_k + \bar{w}_k$ queries. The first batch of u_k queries can be processed by a single replica after $\bar{v}_k$ seconds, the second batch of u_k queries is processed after $2\bar{v}_k$ seconds, and so on. We have a total of $B = (x_k + \bar{w}_k)/u_k$ batches of queries, so the last batch of at most u_k queries is processed after $B\bar{v}_k$ seconds. Hence, at a given time slot k, the query completion time T_k of $x_k + \bar{w}_k$ queries by u_k replicas can be computed by:

$$T_k = \frac{x_k + \bar{w}_k}{u_k}\bar{v}_k \tag{6.22}$$

While this definition of completion time assumes that queries arrive such that the query processors are always busy during the time slot, it behaves as expected: T_k decreases when the number of processing nodes increases, and increases when the number of queued queries, the number of arriving queries or the average query processing time increases.

6.8 Experimental Setup

In the next section, we experimentally investigate to determine the potential of our proposed model for reducing the power consumption of a search engine without negatively impacting on its efficiency.

In particular, five research questions are addressed, as follows:

1. Do our proposed self-adaptive models using seasonal data and current query traffic achieve comparable latency values compared to reasonable baselines while achieving savings in power consumption?

2. Can we improve the modelling of the latency using the deterministic approach instead of queueing theory?

3. How do power and efficiency properties of LONGTERM and SHORTTERM differ?

4. How should the latency cost function be modelled within our self-adaptive models?

5. How does the length of slot T_s impact upon latency and power consumption?

In the remainder of this section, we define the experimental setup to address these research questions, covering the search engine (Section 6.8.1), evaluation measures

Table 6.3: Statistics of the two days of the MSN 2006 query log used within these experiments.

	Query length						Total
	1	2	3	4	5	≥ 5	
Prev. Day	110,706	199,721	136,667	56,183	20,954	10,981	535,212
Curr. Day	115,160	195,483	127,849	53,713	21,411	28,107	545,723

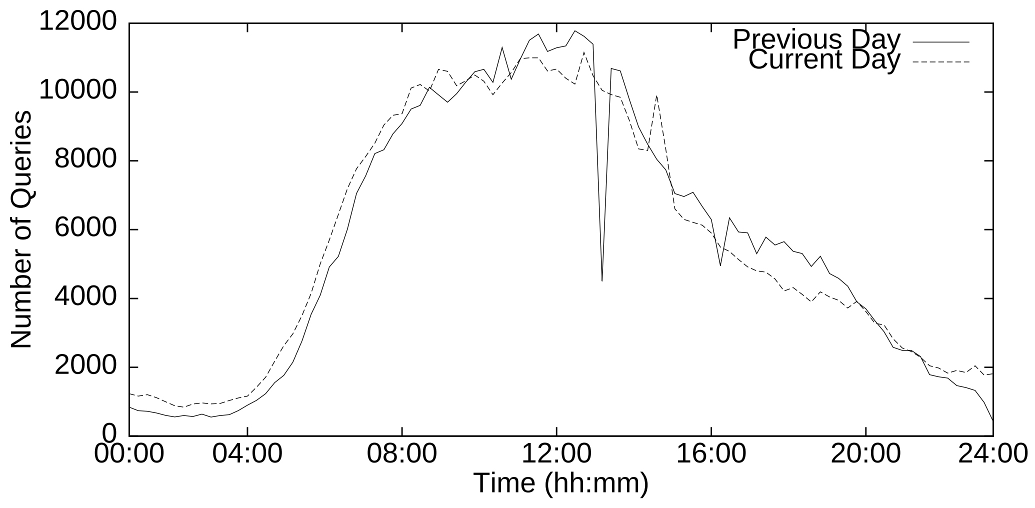


Figure 6.3: Number of queries arriving per 15 minute slot for both days.

(Section 6.8.2), baselines (Section 6.8.3), and parameter settings (Section 6.8.4).

6.8.1 Search Engine, Documents & Queries

To evaluate the proposed model, we determine the processing times for real user queries submitted to a search engine platform built upon a simulation platform. In particular, we index 50M Web documents from the TREC ClueWeb09 corpus (category B) using the Terrier IR platform[3] [78] – ClueWeb09 cat. B is intended to reflect the first tier of a commercial Web search engine index. While indexing the corpus, standard stopwords are removed and Porter stemming applied.

For queries, we use two days of queries (approx. one million queries) from the MSN 2006 query log[4]. In particular, we use queries from two days of two consecutive weeks of May 2006. Figure 6.3 presents the number of queries over the course of

[3]http://terrier.org/

[4]http://research.microsoft.com/en-us/um/people/nickcr/wscd09/

each day and table 6.3 classifies the queries based on its length (number of terms). During retrieval, we use the WAND dynamic pruning technique applying BM25 [85] to rank 1000 documents for each query, recording the processing time of the query by a single replica. All efficiency experiments are made with a quad-core Intel Xeon 2.4GHz, with 8GB RAM and inverted indices stored on a 160GB SATA drive.

6.8.2 Evaluation Measures

As our work concerns balancing the trade-off between search engine efficiency and power consumption, we measure both aspects within our work. In particular, we measure the mean and 90th Percentile (the value below which the 90% of the observations can be found) response time for queries as the configuration of the search engine is varied across the evaluation period (denoted ACT and 90thPC, respectively). Response times are measured in milliseconds (ms).

Concurrently, we measure the power usage of the search engine for the same period based on the number of machines active at any point (denoted E and measured in kWh), as well as the maximum number of machines that were active (denoted Max Mch). We note that with some configurations of the search engine when there are insufficient replicas available, the search engine will become backlogged with excessive number of queued queries. To prevent any skew in the results, we drop queries that are not answered within 5 seconds, thereby returning an error page to the user of that query. Clearly this is an undesirable scenario, and hence, we count the number of unanswered queries (denoted %UQ).

To summarise, we consider as a success when, compared to a baseline approach (described in the next section), the maximum number of active query server replicas and resulting power consumption of the search engine can be reduced, without marked negative impact upon the experience of the search engine users, as portrayed by increased response times and higher rates of unanswered queries.

6.8.3 Baselines

To evaluate our proposed model, we define two reasonable baselines for determining u_k for $i = 0, \dots, N-1$, in other words for defining how many machines are active at any slot.

The first baseline – which we call **Naïve** – is motivated by the provisioning of search engines to cope with worst-case query volume [11], and consists in choosing the maximum number M of machines in each time slot, ignoring completely any power consumption. In this case, we assume maximum power cost, to reflect the fact that the M machines are will continue to be allocated queries, e.g. by round-robin.

The second baseline, that we call **Threshold**, consists in fixing a time threshold for the query completion times, and using the previous day average arrival times, the average processing times and the number of queued queries in the same time slot in Equation (6.22). In doing this, we can derive the decisions u_k for $i = 0, \dots, N-1$ that, *based on the previous day behaviour*, will be applied to the current day. In this case, we assume that each of the selected u_k processing nodes for a specific time slot consume maximum power, as they may be allocated queries in a round robin basis, even if the current query load is lower than the previous day query load. On the other hand, the other $M - u_k$ nodes in a STANDBY state consume $P_{standby}$ power each. Then, if we consider the definition of latency as per Equation (6.22), and fix the time threshold to T^*, we can compute u_k as:

$$u_k = \frac{\bar{w}_k}{T^*}\bar{v}_k$$

where we assume that in each time slot the choice of u_k was able to process all the incoming queries, so that $x_k = 0$. The value of T^* is determined by the length of the time slot, as we want all the queries of a slot to be processed before proceeding to the next slot.

Of these two baselines, it is clear that the power consumption of Threshold will be less than Naïve, as Threshold affords the opportunity to save power for nodes that are not expected to be required. However, compared to LONGTERM, Threshold may disable nodes that would soon be required, as it cannot examine the impact of a power decision across the remainder of the day.

6.8.4 Parameter Settings

To instantiate our model, we invoke various parameter settings as follows. Firstly, to calculate the power consumption of a replicated processing node, we use the energy ratings from the EU Energy Star programme [90] for a small server as follows: $P_{on} = 62W$, $P_{standby} = 2W$, following the reported use of commodity-sized servers within commercial search engines [11]. Within latency costs functions (Equations (6.23)-(6.25)), we follow Wang et al. [94] and use $\alpha = -0.01$ for the ClueWeb09 cat. B corpus. For slot duration, we set $T_s = 15$ minutes, reflecting an interval that identifies general changing trends in query volumes that the model can quickly respond to, rather than random fluctuations that might be detected by shorter slot durations. Finally, as queries arrive on average at 15 per second, we use $M = 15, 20$ as the number of replica query processors. The remaining parameter of our model, namely the power/latency trade-off λ are experimental variables that we vary within the next section.

6.9 Results

In this section, we aim to determine if our proposed self-adapting model allows the system to markedly reduce power consumption with latency comparable to that achieved by the baselines. This section drives us towards the answers to the research questions presented in section 6.8. In particular, Section 6.9.1 firstly determines the efficiency and power properties of our two baselines to present later in Section 6.9.2 the performance of our model using Queueing Theory. Section 6.9.3 finally answers our first, second and third research questions, by comparing LONGTERM and SHORTTERM approaches with the baselines and with the Queueing Theory approaches. In Section 6.9.4, we address our fourth research question concerning the choice of latency cost function within the model. Finally, Section 6.9.5 studies the influence of choice of the slot length T_s .

6.9.1 Baselines

The top part of Table 6.6 reports, for $M = 15, 20$ replicas, the various evaluation measures achieved by the two baselines in this paper, namely Naïve, which keeps the maximal number of replicas active, and Threshold, which uses the number of replicas active that would have sustained the traffic of the previous day. Within the table, we report efficiency measures (average and 90th percentile response times, measured in milliseconds), the number of unanswered queries, the peak number of machines used, and the total energy consumption over the course of the day (kWh). The time slot size is maintained at $Ts = 15$ minutes.

Analysing the response times for the baselines within Table 6.4, we note that mean response times around 850ms are achievable by both approaches. Moreover, while the Threshold approach can reduce the energy consumption compared to Naïve by putting machines into standby mode (by 47% for $M = 15$ and 57% for $M = 20$), this comes at the expense of marginally increased response times (approx. 6ms). To summarise, we find that, as expect the Threshold approach results in markedly decreased energy consumption compared to Naïve, with little marked impact on response times.

Table 6.4: Performance comparison among the Baselines for different M and λ values.

λ	ACT(ms)	90thPC	% UQ	Max Mch	E(kWh)
$M = 15$					
Baselines					
Naïve	847	1,349	0	15	22.3
Threshold	853	1,354	0.03	15	12.0
$M = 20$					
Baselines					
Naïve	846	1,349	0	20	29.8
Threshold	853	1,354	0.03	19	12.9

6.9.2 Queueing Theory aproaches

In section 6.7 we presented two different ways of defining the latency cost function. The first one 6.7.1 is based on Queueing Theory, meanwhile the second one 6.7.2 uses a deterministic approach. This section analyses the performance of the Queueing Theory modelled latency function and its particular behaviour. We compare the results obtained with the two implemented baselines in table 6.5. QT-LONGTERM

Table 6.5: Performance comparison among QT-LongTerm, QT-ShortTerm and the Baselines for different M and λ values.

λ	ACT(ms)	90thPC	% UQ	Max Mch	E(kWh)
$M = 15$					
Baselines					
Naïve	847	1,349	0	15	22.3
Threshold	853	1,354	0.03	15	12.0
QT-LongTerm					
0.25	847	1,348	0	15	11.3
0.5	847	1,348	0	15	11.1
0.75	847	1,349	0	15	10.9
QT-ShortTerm					
0.25	849	1,349	0.02	15	11.2
0.5	849	1,349	0.02	15	11.1
0.75	849	1,350	0.02	15	10.6
$M = 20$					
Baselines					
Naïve	846	1,349	0	20	29.8
Threshold	853	1,354	0.03	19	12.9
QT-LongTerm					
0.25	847	1,348	0	20	14.4
0.5	847	1,348	0	20	13.6
0.75	847	1,349	0	20	13.2
QT-ShortTerm					
0.25	847	1,348	0	20	14.62
0.5	848	1,349	0.02	20	13.7
0.75	849	1,350	0.02	20	13.1

refers to the LongTerm approach based on Queueing Theory and QT-ShortTerm is based on ShortTerm approach.

Comparing QT-LongTerm with Naïve, latency values are substantially equal – 847 ms for $M = 15$ –, but QT-LongTerm achieves an energy saving up to 51%, due to the reduction of powered-on machines on the lowest contention slots of the day. Similar behaviour occurs for $M = 20$. Regarding Threshold, for $M = 15$, the results are quite similar with regard to QT-LongTerm. Looking at $M = 20$ machines, QT-LongTerm improves the average completion time by only a 0.7%, at the cost of increasing the energy consumption by an 11%. This behaviour has an easy explanation: Queueing Theory methods are at a disadvantage regarding Threshold: when the contention of the system increases, configurations with low number of machines become non-stationary ($\rho \geq 1$). This means that the model can not calculate a value for the mean waiting time (W), although some of those configurations would be valid (queries have to wait more time, but in any case they will be solved). In this cases, the system can consider only the configurations with higher number of machines, reaching a point where none of the configurations are stationary, so we select the maximum number of available machines. QT-SHORTTERM evidences analogous results to QT-LONGTERM.

6.9.3 Self-adaptive Power/Latency Models

The bottom two parts of Table 6.6 report the evaluation measures for the LONGTERM and SHORTTERM approaches. For both approaches and both $M = 15, 20$ replicas, we vary the power/latency trade-off parameter λ, to determine its impact on both efficiency and energy consumption. In this section, the L_1 latency cost function (Equation (6.23)) is applied.

Firstly, we discuss the LONGTERM approach, which estimates the expected number of queries solely based on the query volume in the same time slot of the previous day. Overall, for some values of λ, this approach provides completion times generally comparable with the baselines, i.e. less than 900ms. However, such values can be obtained with marked reductions in energy use. For instance, compared to Threshold, the setting of $M = 20, \lambda = 0.5$ produces a 5% increase in mean completion times and 4% in 90th percentile completion time; this is achieved with a 42% reduction in consumed energy, and a peak usage of 7 replicas – down from 19. For $M = 15$, $\lambda = 0.5$ the query load can be serviced with only 6.3 kWh of energy use and only 6 replicas, at the cost of 14% increase in mean completion time compared to Threshold, and a small increase in the number of unanswered queries.

Such results demonstrate the promise of the proposed LONGTERM approach: given enough replicas to service peak demands ($M = 20$), it can achieve marked energy savings compared to the Threshold baseline. Indeed, LONGTERM has the advantage that by being able to derive the cost of a decision until the end of the day, compared to Threshold it has less tendency to overfit to any fluctuations in query volume experienced by the search engine on the previous day.

Next, we examine the results for the SHORTTERM approach in the bottom part of

Table 6.6: Performance comparison among LONGTERM, SHORTTERM and the Baselines for different λ and M values.

$M = 15$					
λ	ACT(ms)	90thPC	% UQ	Max Mch	E(kWh)
Baselines					
Naïve	847	1,349	0	15	22.3
Threshold	853	1,354	0.03	15	12.0
LONGTERM					
0.25	848	1,350	0	15	12.0
0.5	969	1,394	1.35	6	6.3
0.75	3,194	3,886	19.19	2	2.7
SHORTTERM					
0.25	857	1,356	0.03	8	8.0
0.5	1,536	2,260	1.36	4	4.8
0.75	3,225	3,724	11.84	2	2.9

$M = 20$					
λ	ACT(ms)	90thPC	% UQ	Max Mch	E(kWh)
Baselines					
Naïve	846	1,349	0	20	29.8
Threshold	853	1,354	0.03	19	12.9
LONGTERM					
0.25	848	1,349	0	17	12.1
0.5	896	1,409	0.08	7	7.4
0.75	2,299	3,049	4.68	3	4.2
SHORTTERM					
0.25	854	1,354	0.02	10	9.5
0.5	1,007	1,567	0.31	5	5.7
0.75	2,956	3,700	10.69	2	3.4

Table 6.6. Recall, as explained in Section 6.5, that compared to LONGTERM, SHORTTERM also takes into account the actual number of arrived queries during the previous slot, while LONGTERM only considers the query traffic from the previous day. In general, we find that for $\lambda = 0.25$, SHORTTERM exhibits an improved power/latency trade over the results exhibited by the LONGTERM approach. In particular, with mean response times that are only a few milliseconds different from the results of the Threshold and Naïve baselines, SHORTTERM achieves marked energy savings (33% and 64% respectively for $M = 15$; 26% and 68% for $M = 20$).

This further marked reduction in energy consumption shows that the SHORTTERM method can more accurately predict the query load in the next slot by considering the query load in the previous slot. This way, as LONGTERM makes the estimation based on previous query volume alone, it selects a higher number of machines, while SHORTTERM can reduce the necessary number of replicas, with corresponding energy savings. This is clearly illustrated in Figure 6.4, which shows the number of machines used by each approach for $M = 20$ and $\lambda = 0.5$, as well as the two baselines. LONGTERM and SHORTTERM clearly reduce the number of active machines, but are less sensitive than Threshold to the sudden decrease in query traffic at approx. 13:00 on the training day. Moreover, SHORTTERM uses less query processors than LONGTERM over the day, confirming the power results in Table 6.6.

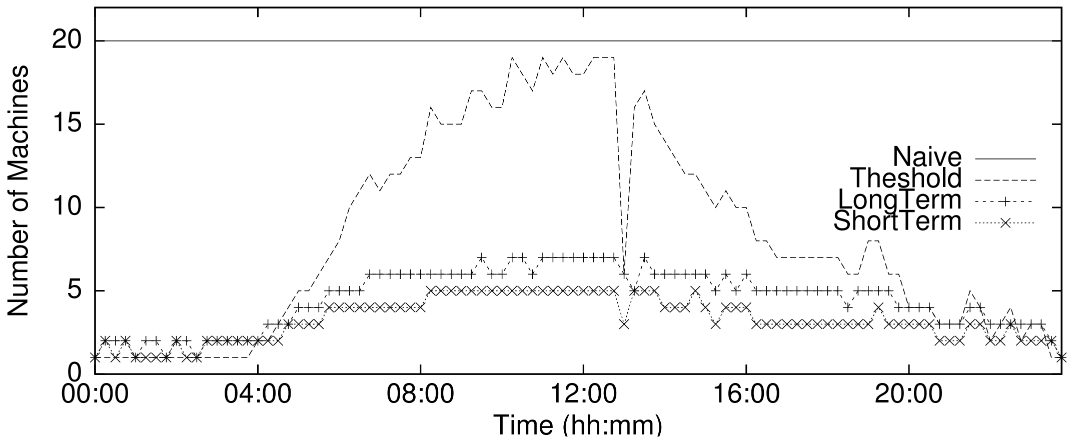


Figure 6.4: Number of machines used along the day by LONGTERM and SHORTTERM ($\lambda = 0.5$, $M = 20$) and the two baselines, namely Naïve and Threshold.

In summary, in answer to our first research question, we find that our proposed self-adapting models for adapting the number of available query processors to query demand can markedly reduce power consumption, without marked impact on efficiency.

Regarding our second research question, we have attested that the deterministic approach is more suitable to represent the latency of a search engine than the $M/M/s$ model.

With respect to the third research question, of the proposed LONGTERM and SHORTTERM instantiations, SHORTTERM demonstrates the highest promise, as by considering recent query traffic conditions, it is able to reduce the number of required query processors, without marked degradations in query response time. Indeed, with

$\lambda = 0.25$ and $M = 15$ SHORTTERM achieves a 33% of power improvement by producing only a 1% increasing in latency; with $\lambda = 0.5$ and $M = 20$ SHORTTERM achieves around mean completion times of 1 second but attains a 24% power saving and maintains the percentage of unanswered queries under 0.05%. In the remainder of this section, we analyse the impact of some parameters of the model: the choice of latency cost function and time slot length on the achieved efficiency and power consumption. As the performance of LONGTERM and SHORTTERM is more desirable than QT-LONGTERM and QT-SHORTTERM, next sections are focused on the deterministic approaches.

6.9.4 Modelling Latency Costs

In this section, we address our fourth research question, examining how the latency of the search engine should be modelled within our proposed approach. Firstly, recall that Wang et al. [94] proposed three different efficiency metrics. In particular, the latency of the search engine can be modelled with an exponential decay function (Equation (6.23), denoted L^1), a step function with a fixed penalty after a time threshold has expired (Equation (6.24), L^2) or a step function followed by an exponential decay (Equations (6.25), L^3). Figure 6.5 illustrates each of the latency cost functions. The exponential parameter $\alpha > 0$ controls how rapidly the latency cost increases as a function of query completion time. As we consider the processing of queries within a time slot, we set the time threshold as the slot length $T_s = 15$ minutes.

$$L_k^1(\cdot) = 1 - \exp(\alpha T_k) \tag{6.23}$$

$$L_k^2(\cdot) = \begin{cases} 0 & \text{if } T_k \leq T_s \\ 1 & \text{otherwise} \end{cases} \tag{6.24}$$

$$L_k^3(\cdot) = \begin{cases} 0 & \text{if } T_k \leq T_s \\ 1 - \exp(\alpha(T_k - T_s)) & \text{otherwise} \end{cases} \tag{6.25}$$

While all metrics consider processing of all queries in the time slot, in contrast to L_k^1, the L_k^2 and L_k^3 model a certain tolerance level for query execution time, based on the processing of all queries in the time slot.

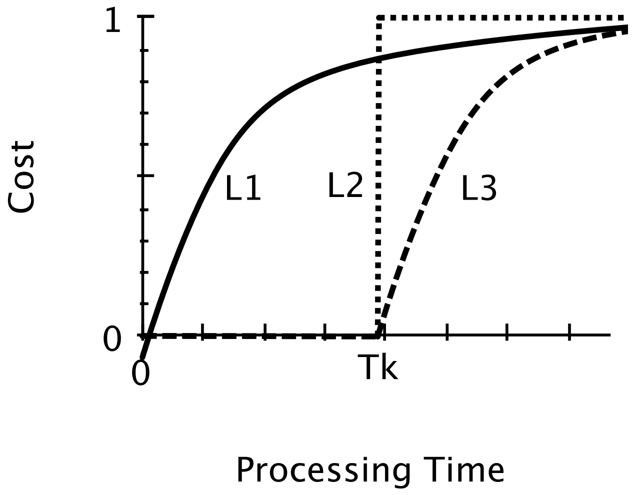


Figure 6.5: Illustration of latency cost function shapes.

Table 6.7: Comparison of latency functions, while varying λ.

	ACT(ms)	90thPC	% UQ	Max Mch	E(kWh)
$\lambda = 0.25$					
L^1	848	1,349	0	17	12.1
L^2	869	1,380	0.04	20	14.2
L^3	880	1,401	0.07	19	12.0
$\lambda = 0.5$					
L^1	896	1,409	0.07	7	7.4
L^2	849	1,350	0.01	20	14.2
L^3	860	1,357	0.03	7	7.6
$\lambda = 0.75$					
L^1	2,299	3,049	4.68	3	4.2
L^2	3,544	4,113	22.77	9	3.9
L^3	2,235	3,030	4.5	3	4.3

Table 6.7 shows the comparison between the chosen latency functions for the LongTerm approach for $M = 20$. We also vary λ, in case the choice of λ can impact upon the choice of the latency function. On a first inspection of Table 6.7, we observe that latency function L^1 achieves the lowest response times and energy consumptions. This can be explained with reference to Figure 6.5: For L^2, the cost function

Table 6.8: Effect of T_s using LongTerm

Ts (min)	ACT(ms)	90thPC	% UQ	Max Mch	E(kWh)
LongTerm					
5	1,551	2,276	1.28	5	4.95
15	896	1,409	0.08	7	7.425
30	851	1,351	0.01	9	9.39
60	1,894	1,780	12.38	13	8.7

becomes 1 as soon as queries cannot be completed on time. Hence, if the latency exceeds the time threshold even by a small amount, the latency cost is 1, and the model resorts to the power function to decide between options. As it is power conservative in nature, a smaller number of machines will be chosen. In contrast, by using the exponential decay, L^1 and L^3 represent 'softer' latency cost function, and hence can permit small inefficiencies for power savings. Of these, the simpler L^1 is more appropriate than L^3, for the same reasoning as for L^2. In summary, in addressing our fourth research question concerning how the latency cost should be modelled, we find that the exponential increase of cost as latency increases represents the most promising function, as per Equation (6.23).

6.9.5 Effect of time slot length

We now address the final research question concerning the effect of time slot length on the efficiency and power consumption. Table 6.8 shows the results obtained with LongTerm using different lengths for the time slot T_s: 5, 15, 30 and 60 minutes.

On analysing the table, we can confirm the expected behaviour. Small T_s values such as 5 minutes can be altered by small peaks of traffic that would not have importance at all if they would be considered on a larger interval. In addition, this lead to suppose the continue changing of the state of the query servers, with consequent reconfiguration time loss.

On the opposite case, 60 minutes interval is probably avoiding some important peaks as the one of the previous day after midday (see Figure 6.3). 15 and 30 minutes intervals are more approapiate to this kind of scenario, as they bring the opportunity to react to important changes on query traffic and avoid changes the system due to trifling peaks of queries.

6.10 Model Generalization

This chapter has presented a new latency-power trade-off model for a single-shard configuration (see Figure 6.1). Note that the scheduling is performed by using a FIFO approach and none of the methods we have presented in previous chapters, such as Queue Length, Least Loaded or the Hybrid approach.

The aim of this section is to demonstrate that the presented model can easily be adapted to more than one shard and different scheduling methods.

The general idea is to apply Equation 6.6 on every shard, from 1 to S, being S the total number of shards.

Consider Equations 6.10 and 6.22. Power function (Eq. 6.10) depends on u_k (the number of active replicas on slot k), and nothing changes regarding one-shard architecture. If we study the latency function and its deterministic approach (Eq. 6.22), we must care about the following parameters:

- x_k: number of queries waiting in the queues at the beginning of slot k. Up until now we have considered a unique FIFO queue for all the replicas of one shard. Some other scheduling methods can be used, as the ones we work with in Chapter 4. These scheduling methods (Round Robin, Queue Length, Least Loaded and also Hybrid approach) work with one queue on each replica, and not with a general queue. If we implement these models, we must sum up the number of queries waiting on each active replica x_{kr}:

$$x_k = \sum_{r=1}^{u_k} x_{kr} \tag{6.26}$$

- $\bar{w}_k$: number of queries that will arrive during slot k. This parameter has the same value for all the shards, due to the document partitioning approach. Nothing changes regarding presented one-shard architecture.

- $\bar{v}_k$: average processing time of queries during slot k. Nothing changes regarding presented one-shard architecture, each shard have its own $\bar{v}_k$ value that can be estimated based on previous query log.

- u_k: number of machines (replicas) used during slot k. Once more, nothing changes regarding presented one-shard architecture (considering the same number of replicas for all shards).

An important problem of considering distributed queues is how to proceed with the queries that are enqueued if we decide to switch-off/standby its replica [59]. Two options are possible:

1. Re-scheduling the queries over the remained active replicas.
2. Changing the power state of the replica when it finishes processing the enqueued queries. If the enqueued workload is low, this option is worth, otherwise the first approach is more suitable.

6.11 Conclusions

While various research efforts have been dedicated to reduce power consumption regarding IT systems, few works apply this concept to the Information Retrieval field. In this chapter we have proposed a mathematical model for a replicated search engine that allows the establishment of a trade-off between latency and power consumption. Based on the query traffic from a previous day, the system predicts the incoming query flow and increases, decreases or maintains the number of available replicated query processors to answer the queries under acceptable latencies. Experiments are conducted in comparison to two baselines: Naïve (always uses the maximum number of machines) and Threshold (establishes a maximum latency value and calculate the number of necessary machines to ensure latency values under a threshold), using the processing times of 1 million queries submitted to a real commercial search engine. Our results show that our proposed self-adapting model can achieve an energy savings of 33% while only degrading mean query completion time by 10 ms compared to a baseline that provisions replicas based on a previous day's traffic, while more substantial energy savings can be attained while accepting marginally larger efficiency degradations.

We focused on the power savings achievable when switching replicated query servers between standby and actively processing search queries. There are advantages to such a scenario, because during off-peak times standby query servers may be re-purposed to other offline tasks, such as indexing, ads/recommendations generation, and pre-caching of result lists [52].

Chapter 7

Conclusions and Future Work

This chapter discusses the conclusions achieved from this thesis and proposes some directions on how to continue researching on each addressed topic.

7.1 Conclusions

7.1.1 Simulation platforms are reliable for large-scale IR experimentation, leading to resource savings.

Chapter 3 mainly constitutes a survey of the main evaluation platforms that have been used to test the efficiency of search engines. The choice depends on different factors we detailed, as the difficulty of mapping the system into an analytical or simulation model, the dimensions of the architecture or the financial and physical resources at our disposal. After studying in depth all the approaches we concluded that simulation constitutes a good alternative for representing an IR system, due to its accuracy in representing a real search engine. Besides, it contributes to the Green IR behaviour, as it allows high energy and resource savings. This conclusion has encouraged us to use simulation for our experimentation.

7.1.2 Query efficiency predictors improve query scheduling.

Few published works are based on query scheduling methods. Round Robin, Random or First-In-First-Out are the most used approaches. Nevertheless, on Chapter 4 we have detailed how these methods consider that all the queries have the same response time. After explaining the widely used dynamic pruning techniques, it was easy to demonstrate that the previous assumption about the queries was really far form reality. We proposed a new method, namely Least Loaded, that uses query efficiency predictors for summing up the processing times of the previously enqueued queries. This way, we calculate the estimated time that a query must be waiting before being processed on each query server. We select the replica with the lowest estimated waiting time. Experiments were driven using different query sets (synthetic and real data and different query traffic conditions) and several distributed architectures with different number of shards and replicas. We have demonstrated that query efficiency predictors are able to reduce query waiting time of queries and increase the efficiency of large-scale search engines.

7.1.3 Hybrid scheduling methods avoid the overhead inherent to query predictors and improve the state-of-the-art approaches.

Chapter 5 identified the drawback of Least Loaded and proposed a new hybrid method that gathers the benefits of Least Loaded and combines it with the lighter Queue Length and Round Robin approaches. Experiments using ClueWeb category B dataset, different query traffic conditions as well as several architectures, have demonstrate the power of this approach to increase the efficiency of the system.

7.1.4 Our new power/latency trade-off mathematical model contributes to achieve high energy savings without compromising the efficiency of the system.

The mathematical model presented in Chapter 6 is the main contribution of this thesis. We proposed a self-adapting model for large-scale search systems that establishes a trade-off between latency and power consumption in terms of the number of replicated query servers required as query load varies throughout the day. At the beginning and at the end of the day, the system reduces automatically the number of active machines in order to save energy, meanwhile during midday the number of active machines reaches its maximum. We showed how this model can be instantiated for different methods of forecasting the query traffic at a given time – based on current and historical query loads – as well as with a variety of latency functions. We thoroughly demonstrated experimentally how the proposed model can reduce the power consumption of a search engine by 33% with little decrease in the overall efficiency of the search engine. We generalized the proposed one-shard model for a multiple-shard architecture and different scheduling methods.

This model is a contribution to the *Green IR* behavior that is being increasingly introduced by big companies. Either if the system turns off/standby the machines or uses them for solving other tasks, it contributes to increase financial savings in IR companies.

7.1.5 $M/M/s$ Queueing Theory model is not suitable for representing the latency of a large-scale search engine.

Chapter 6, and more specifically, Section 6.7.1, applied the $M/M/s$ queueing theory model to estimate the waiting time that a queue must be waiting on a queue. We wanted to check the suitability of this kind of models to implement the latency function into the power/latency trade-off model. Our experiments showed that when the contention of the system is low, its performance is similar to the baselines. Nevertheless, when the incoming query traffic rises, the system becomes not-stationary, and $M/M/s$ is not able to provide a solution. We decided to use the maximum number of available machines in the system, but this approach does not achieve any improvement regarding the *Threshold* baseline. The deterministic approach used instead solved this problem.

7.2 Directions for Future Work

This section discusses several directions for future work related to, or stemming from this thesis. We categorize them into three subsections: IR evaluation platforms, query scheduling techniques and power/latency trade-off model.

7.2.1 IR Evaluation Platforms

The experiments carried out in Chapter 3 regarding virtualization, only tested the power of KVM, a widely used virtualization platform, in order to study its accuracy regarding real systems. Nevertheless, we consider that it would be interesting to experiment more virtualization platforms. They usually offer different configuration options with diverse fitting parameters. This way, maybe we could find a more suitable virtualization platform to represent a real scenario.

7.2.2 Query Scheduling

In order to make our method more realistic it would be interesting to consider *caching*. In our experiments, we have considered that a query is always enqueued into a query server to be solved, but we have not implemented the caching of queries, that allows to answer them without being sent to the query server, just by using the results stored from a previous search of the same query.

7.2.3 Power/Latency Trade-off Model

We have represented an scenario where the servers are powered on and standby depending on the incoming query traffic rates. We have considered that a server does not spend time in turning itself on from standby state. For future work, we will consider a more complex scenario where servers are fully powered down when not required, but incur delay on startup.

It would be also interesting to make a financial savings study, such as in some other published works as [86] and [53]. The power cost can be mapped to financial cost in terms of electricity price and the cost functions can take various properties of this cost into account, such as temporal or regional variations of the electricity price.

Including hardware parameters in the model would make it even more realistic. Variables such as memory or CPU usage are interesting factors to be considered when making the decision of changing the state of a query server. Nevertheless, this would lead to increase the complexity of the model.

In accordance with the Regulations of the Ph.D. studies passed by the Governing Council of the University of A Coruña at its meeting of July 17*th* *2012, it is reproduced below a summary of this thesis in Spanish.*

Appendix A

Resumen

La web se ha convertido en el mayor repositorio de información de todos los tiempos. Los motores de búsqueda actuales deben enfrentarse y saber responder adecuadamente ante este veloz incremento de información y a un enorme y dinámico tráfico de consultas. El éxito y los ingresos recibidos por parte de las empresas de Recuperación de Información (RI) Web dependen de la rapidez de respuesta de las consultas que reciben y de la calidad de los resultados ofrecidos. Para controlar esta situación, las grandes compañías se han visto obligadas a construir grandes centros de datos, geográficamente distribuidos, y compuestos por miles de servidores. El suministro eléctrico de estas inmensas infraestructuras supone un enorme gasto energético, y una pequeña mejora a nivel de eficiencia puede suponer grandes ventajas económicas.

Esta tesis representa una nueva aportación al estado del arte actual referido a gestión de consultas y consumo energético de grandes centros de datos, lo que permitirá a grandes compañías de Recuperación de Información la construcción de motores de búsqueda dotados de mayor eficiencia y su integración en el concepto de *Green Information Retrieval*.

Por una parte, esta tesis propone nuevas técnicas de distribución de consultas a los servidores que las procesan para disminuir su tiempo de respuesta. Mediante técnicas de predicción del tiempo de ejecución de las consultas que están en cola, esperando a ser procesadas por un servidor, es posible estimar cuál será el que procesará la consulta con el menor tiempo de espera.

Por otra parte, esta tesis define un modelo matemático simple que establece un balance entre la latencia (tiempo de respuesta) que ofrece un motor de búsqueda y el consumo eléctrico que genera. Habitualmente estos son parámetros opuestos, de modo que intentar mejorar uno de ellos va en detrimento del otro. Las fluctuaciones de tráfico de consultas a lo largo de todo el día son la clave en torno a la que gira este modelo: cuando el sistema recibe un número de consultas muy elevado, que puede ocasionar mayores tiempos de espera, el modelo automáticamente incrementa el número de máquinas activas en el sistema para mantener unos valores de latencia adecuados. Del mismo modo, cuando la carga del sistema es baja, el modelo reduce el número de servidores activos, lo que genera grandes porcentajes de ahorro energético, especialmente en intervalos horarios de baja actividad.

Experimentos sobre diferentes conjuntos de datos, diversas tasas de tráfico e historiales de consultas reales y sintéticos, atestiguan el gran poder, tanto de los métodos de distribución de consultas como del modelo matemático, para lograr grandes mejoras en cuanto a eficiencia y a ahorro energético, con respecto a los métodos de referencia.

A.1 Motivación

La investigación de esta tesis está motivada por diferentes carencias en el campo de la Recuperación de Información, para las que esta tesis propone diversas soluciones.

En primer lugar, no existe un copioso estado del arte en lo que se refiere a la gestión de consultas en los motores de búsqueda a gran escala. Con gestionar una consulta nos referimos a enviarla al servidor adecuado para que la procese y genere los resultados correspondientes. Las aproximaciones ya existentes antes de la realización de esta tesis son verdaderamente sencillas, pero asumen condiciones inadecuadas, lo que genera una gestión ineficiente con un correspondiente aumento en el tiempo de respuesta de las consultas.

La preocupación por reducir el tiempo de respuesta ofrecido por un motor de búsqueda, hace que una de las soluciones adoptadas sea ampliar el número de servidores que dan respuesta a las consultas de los usuarios. Esta medida, sin embargo, incrementa considerablemente el consumo energético, lo que afecta no sólo a los costes económicos de las empresas sino también a las emisiones de CO_2. Actualmente, algunas empresas de RI están ya dedicando incontables recursos para disminuir su consumo energético: situación estratégica de centros de datos para un mayor aprovechamiento de energías renovables, reutilización de recursos, sistemas más eficientes... Varias empresas como Google o Microsoft incluso publican sus emisiones de carbono y muestran las medidas adoptadas para dar ejemplo a otras empresas que se quieran unir a este comportamiento ya acuñado como *Green IR - Green Information Retrieval*. El control del cambio climático y la sostenibilidad pasa por expandir estos hábitos a un mayor número de empresas del sector tecnológico.

Centrándonos en los motores de búsqueda, el tráfico de consultas que reciben varía considerablemente a lo largo del día, habiendo períodos de baja carga al inicio o final del día y otros períodos de alto tráfico en las horas centrales. De este modo, parte de los servidores que procesan consultas cuando el tráfico es elevado, no son utilizados en períodos de baja actividad, por lo que podrían ser apagados, puestos en suspensión o bien asignados a otras tareas.

Existen trabajos en el campo de la RI estudiando la eficiencia energética de los motores de búsqueda. Sin embargo, hasta donde alcanza nuestro conocimiento, no existen trabajos previos que se encarguen de activar o desactivar servidores de un motor de búsqueda en función de la carga de trabajo de un centro de datos.

De este modo, si en primer lugar reducimos el tiempo de respuesta de las consultas, mejorando los algoritmos de gestión de las mismas, y en segundo lugar reducimos el consumo energético de los motores de búsqueda, podremos conseguir sistemas más eficientes con su correspondiente ahorro económico, lo que constituirá un gran incentivo para las empresas de Recuperación de Información.

A.2 Objetivos

El principal objetivo de esta tesis lo constituye el estudio exhaustivo que permita corroborar los dos enunciados o hipótesis siguientes:

- La eficiencia de un sistema de Recuperación de Información a gran escala puede ser mejorada usando predictores del tiempo de ejecución de las consultas a la hora de distribuirlas a los servidores adecuados para generar su respuesta.

- El consumo energético de un motor de búsqueda a gran escala puede reducirse sin comprometer para ello la eficiencia del sistema.

Ambos objetivos están dirigidos por la eficiencia del motor de búsqueda. El primero de ellos persigue directamente disminuir el tiempo de respuesta que el usuario debe esperar a que su consulta sea resuelta. El segundo punto de interés persigue disminuir el consumo energético de un motor de búsqueda, manteniendo siempre presentes los tiempos de respuesta de las consultas y reaccionando en base a ellos.

A.3 Estructura

Las principales contribuciones de esta tesis se presentan en los Capítulos 3, 4, 5 y 6. El Capítulo 2 introduce algunos conceptos de RI para los lectores no expertos en el campo. La organización de los capítulos de la tesis es como sigue:

- El Capítulo 2 presenta conceptos básicos de RI sobre los que se fundamenta esta tesis. En particular, se definen conceptos que componen el proceso general de Recuperación de Información, como son el proceso de indexación y *crawling*. Del mismo modo, indicamos cómo los sistemas de RI han evolucionado gracias al uso tan extendido de la Web, confirmando los sistemas de RI web. Estos sistemas han obligado la construcción de grandes sistemas distribuidos, ante la imposibilidad de almacenar toda la extensa colección de material on-line en un único servidor. Estas grandes plataformas dan pie a la definición del término *Green IR*, tratado en una de las secciones de este capítulo. El concepto de evaluación de sistemas de RI pone fin a este capítulo, dando paso al Capítulo 3 donde se tratará más en detalle.

- El Capítulo 3 es un estudio de las principales plataformas de evaluación de sistemas de Recuperación de Información: sistemas reales, virtualización o simulación, entre otras. En este capítulo se realiza una comparación considerando, no sólo costes financieros, sino dificultad en el desarrollo de cada una de las aproximaciones o incluso la fiabilidad de los resultados. Desarrollamos además una parte experimental con el objetivo de comparar la idoneidad de las plataformas de evaluación a la hora de representar un sistema de Recuperación de

Información. Este capítulo nos permite tomar la decisión acerca de la plataforma empleada para realizar los experimentos que se detallan en los capítulos siguientes.

- El Capítulo 4 propone un nuevo método, llamado *Least Loaded*, que mejora las técnicas de gestión de consultas presentes en el estado del arte. Esta técnica se basa en predecir el tiempo que una consulta estará esperando en cada uno de los servidores disponibles, y dirigirla al que ofrezca un menor tiempo de espera. Para ello usamos unos predictores que demuestran ser unos estimadores muy potentes para nuestro caso de estudio. Se presentan resultados experimentales usando dos conjuntos de datos con el objetivo de analizar el funcionamiento del método propuesto bajo diferentes condiciones como pueden ser la arquitectura del motor de búsqueda (número de servidores y réplicas) y el tráfico de consultas.

- El Capítulo 5 analiza las desventajas del método propuesto previamente, y las combate proponiendo un método híbrido de gestión de consultas que adopta el comportamiendo de métodos sencillos cuando la carga del sistema es lo suficientemente ligera como para necesitar un método tan potente como *Least Loaded*, e imita a éste último en situaciones en las que los servidores se saturarían con los métodos tradicionales y los tiempos del respuesta no serían aceptables. Esta aproximación permite mejorar el comportamiento de las técnicas anteriores, y con ello, el estado del arte.

- El Capítulo 6 está motivado principalmente por el concepto de *Green IR*, que se refiere a la sostenibilidad energética en grandes centros de Recuperación de Información. En este capítulo definimos un modelo matemático que establece un balance entre la latencia obtenida por el sistema y el consumo energético que genera, con el objetivo de desactivar servidores cuando la carga del sistema permita que estén libres de trabajo. De este modo, el modelo automáticamente enciende o pone en suspensión a los serviores, dependiendo del tráfico entrante. Experimentos con flujos de consultas reales permiten comprobar el gran ahorro energético con respecto a los modelos de referencia y cómo nuestro modelo mantiene los porcentajes de latencia.

- El Capítulo 7 cierra esta tesis enumerando las conclusiones derivadas de este trabajo, así como posibles líneas de trabajo que pueden ser acometidas en un futuro.

A.4 Conclusiones

A continuación exponemos las principales conclusiones obtenidas de este trabajo de investigación.

A.4.1 Los modelos de simulación son plataformas fiables para desarrollar experimentos a gran escala de Recuperación de Información, consiguiendo un gran ahorro de recursos.

El Capítulo 3 constituye un estudio de las principales plataformas de evaluación usadas en el campo de la Recuperación de Información a la hora de estudiar la eficiencia de motores de búsqueda. La elección de la plataforma adecuada depende de diferentes factores que hemos estudiado, tales como la complejidad de representación del modelo real en un modelo analítico o simulado, las dimensiones de la arquitectura elegida o bien de los recursos físicos y económicos que tengamos a nuestra disposición. Después de estudiar en profundidad todas las aproximaciones, hemos concluido que las plataformas de simulación constituyen una buena alternativa para representar sistemas de RI, debido a que permiten representar con alta fiabilidad un sistema real. Además, al permitir ahorro de recursos, con su correspondiente ahorro energético, contribuye a los principios incluidos en *Green IR*. Este estudio nos ha impulsado a usar simulación como nuestra plataforma de experimentación.

A.4.2 La predicción del tiempo de ejecución de las consultas mejora la eficiencia de los motores de búsqueda.

Pocos son los trabajos encargados de estudiar cómo enviar las consultas a los servidores que las procesan. *Round Robin* o *First-In-First-Out* son las aproximaciones más usadas. Sin embargo, en el Capítulo 4 explicamos cómo estos métodos consideran que todas las consultas tienen el mismo tiempo de respuesta. Tras explicar las ampliamente usadas técnicas de *pruning* o poda, ha sido fácil demostrar cómo esas suposiciones no eran correctas. Hemos propuesto un nuevo método, llamado *Least Loaded*, que usa predicción del tiempo de ejecución de consultas (*query efficiency predictors*) para calcular el tiempo de procesado de todas las consultas que están en cola esperando a ser procesadas. De este modo, podemos estimar el tiempo que una consulta debe esperar en cada uno de los servidores de consultas antes de ser procesada. Así, se seleccionará el servidor que ofrezca un menor tiempo de espera. Para comprobar esta aproximación hemos utilizado conjuntos de datos tanto sintéticos como reales, con tráfico variable, y diferentes arquitecturas distribuidas con distinto número de servidores y réplicas. Con estos experimentos hemos podido demostrar que predecir la eficiencia de las consultas nos permite reducir en un alto porcentaje el tiempo de espera de las consultas, lo que supone mejorar la eficiencia de los grandes motores de búsqueda.

A.4.3 La combinación de métodos de gestión de consultas evita la pequeña sobrecarga de las técnicas de predicción y mejora las aproximaciones existentes.

El capítulo 5 identifica la desventaja de usar predictores de la eficiencia de las consultas: el tiempo empleado en el cálculo de dichas predicciones supone un pequeño retraso en el sistema, que puede aumentar los tiempos de espera cuando el tráfico de consultas puede ser manejado por métodos más simples como *Queue Length* o *Round Robin*. Por ello, hemos propuesto un método de gestión de consultas híbrido que aprovecha las ventajas de la predicción de consultas cuando la carga del sistema es alta y las combina con la simplicidad de otros métodos como *Round Robin* cuando el tráfico es bajo. Los experimentos, realizados sobre la colección ClueWeb categoría B de 50 millones de documentos y usando tráfico de consultas variable y diferentes arquitecturas del motor de búsqueda, han demostrado la potencia de esta aproximación híbrida para mejorar la eficiencia del sistema.

A.4.4 El modelo matemático propuesto, consigue obtener un balance entre consumo energético y latencia de un motor de búsqueda, proporcionando un alto ahorro energético.

El modelo matemático presentado en el Capítulo 6 es la principal contribución de esta tesis. Hemos propuesto un modelo autoadaptativo para sistemas de búsqueda a gran escala, que permite establecer un balance entre la latencia y el consumo energético generado. Para ello, según la carga de consultas va variando a lo largo del día, también lo hace de un modo proporcional y automático el número de serviores requeridos para responder a esas consultas. Al inicio y al final del día, el sistema reduce automáticamente el número de máquinas activas con el objetivo de ahorrar energía eléctrica. Por el contrario, en escenario de alta carga, como puede ser en las horas centrales del día, el número de máquinas activas puede llegar a alcanzar su máximo. El modelo estima el número de máquinas a usar en función de la carga de consultas en días anteriores o incluso considerando el tráfico actual. El modelo acepta diferentes fórmulas para el cálculo de la latencia del sistema, parámetro que tiene en cuenta a la hora de tomar la decisión de activar o desactivar servidores. Los experimentos han demostrado como el modelo permite alcanzar un ahorro energético de un 33 % sin apenas degradar la latencia del sistema.

Este modelo contribuye a los principios de *Green IR*, que poco a poco está siendo introducido en las empresas de Recuperación de Información.

A.4.5 El modelo $M/M/s$ de Teoría de Colas no es adecuado para estimar la latencia de un motor de búsqueda a gran escala.

El Capítulo 6, y más específicamente, la Sección 6.7.1, aplica el modelo $M/M/s$ de Teoría de Colas para estimar el tiempo que una consulta debe esperar en la cola de un servidor para ser procesada. El objetivo era comprobar la adecuación de este modelo de Teoría de Colas para estimar el valor de la latencia dentro del modelo matemático del Capítulo 6. Los experimentos han demostrado que, cuando la carga del sistema es baja, permite alcanzar un funcionamiento similar al de los métodos de referencia. Sin embargo, ante gran afluencia de tráfico de consultas, el sistema se vuelve no estacionario, y el modelo $M/M/s$ es incapaz de computar una solución.

En estos casos, la solución adoptada ha sido utilizar el máximo número de máquinas disponibles en el sistema, pero esta aproximación no ofrece ahorro energético con respecto al método de referencia *Threshold*. Alternativamente, se ha propuesto una solución determinista que solventa esta limitación del modelo $M/M/s$ para motores de búsqueda.

A.5 Futuras líneas de investigación

Esta sección expone las posibles líneas de trabajo que esta tesis deja abiertas para futuros desarrollos. Estructuramos estas líneas en tres secciones: Plataformas de Evaluación de RI, Gestión de Consultas y Modelo de Balance Latencia/Consumo Energético.

A.5.1 Plataformas de Evaluación de RI.

Los experimentos llevados a cabo en el Capítulo 3 con respecto a las plataformas de virtualización, se han realizado sobre KVM, una plataforma de virtualización ampliamente usada, con el objetivo de estudiar la fiabilidad de las plataformas de virtualización para representar sistemas reales de RI. Sin embargo, consideramos que sería interesante probar el comportamiento de otras plataformas de virtualización que permitan mayor flexibilidad a la hora de configurarlas, para así poder ajustar algunos parámetros que la plataforma usada no permitía configurar. De este modo, podríamos encontrar una plataforma de virtualización que representase más fielmente un escenario real.

A.5.2 Gestión de Consultas.

Con el objetivo de hacer más realistas los métodos de gestión de consultas propuestos en esta tesis, sería interesante considerar el uso de una caché de consultas. En nuestros experimentos, hemos considerado que una consulta siempre se enviaba a un servidor para ser procesada. Sin embargo, si hiciésemos uso de una caché, permitiría responder algunas consultas sin necesidad de enviarlas a los servidores, usando los resultados almacenados de consultas iguales que se han respondido previamente.

A.5.3 Modelo de Balance Latencia/Consumo Energético.

En el escenario representado en el Capítulo 6, se ha considerado que los servidores se activan o pasan a modo suspensión, dependiendo de la tasa de tráfico entrante. El tiempo que un servidor ocupa en activarse si está en suspensión, se ha despreciado. Una importante línea de investigación sería considerar un escenario más complejo en el que los servidores fuesen totalmente apagados cuando no fuese necesario su uso. Este caso requeriría considerar el tiempo que un servidor tarda en ser encendido, lo que puede generar retardos importantes que habría que gestionar, pero que permitirá un mayor ahorro energético.

Un posible estudio que completaría el actual sería añadir costes financieros al modelo, como lo han hecho en otros trabajos publicados ([86] y [53]). Las funciones de coste podrían definirse de un modo más complejo para considerar varias propiedades importantes como variaciones de coste espacio-temporales.

Incluir parámetros hardware en el modelo lo haría incluso más realista. Variables como el uso de memoria o de CPU son factores interesantes a considerar para activar o desactivar un servidor.

A.6 Publicaciones

La mayor parte del contenido de esta tesis ha sido ya publicado, dejando de este modo patente la contribución de este trabajo al ámbito científico de la Recuperación de Información:

- Los contenidos desarrollados en el Capítulo 3 sobre el estudio comparativo de diferentes plataformas de evaluación ha sido publicado en: *Analysis of performance evaluation techniques for Large Scale Information Retrieval. Ana Freire, Fidel Cacheda, Vreixo Formoso and Víctor Carneiro. In Proceedings of LSDS-IR 2013.*

- El método *Least Loaded* de gestión de consultas presentado en el Capítulo 4 ha sido publicado en uno de los más prestigiosos congresos de Recuperación de Información (A*): *Scheduling Queries Across Replicas. Ana Freire, Craig Macdonald, Nicola Tonellotto, Iadh Ounis and Fidel Cacheda. In Proceedings of SIGIR*

2012. (36.5 % aceptación).

- El método híbrido de gestión de consultas estudiado a lo largo del Capítulo 5 dio lugar a la siguiente publicación: *Hybrid query scheduling for a replicated search engine. Ana Freire, Craig Macdonald, Nicola Tonellotto, Iadh Ounis and Fidel Cacheda. In Proceedings of ECIR 2013*. (29 % aceptación).

- Los contenidos que engloba el Capítulo 6 han sido recientemente publicados: *A Self-Adapting Latency/Power Tradeoff Model for Replicated Search Engines. Ana Freire, Craig Macdonald, Nicola Tonellotto, Iadh Ounis and Fidel Cacheda. In Proceedings of WSDM 2014*. (18 % aceptación).

In accordance with the Regulations of the Ph.D. studies passed by the Governing Council of the University of A Coruña at its meeting of July 17*th 2012, it is reproduced below a summary of this thesis in Galician.*

Appendix B

Resumo

A Web converteuse no maior repositorio de información de todos os tempos. Os actuais motores de busca deben enfrontarse e saber responder axeitadamente ante este cada vez maior incremento de información e á consecuente demanda dun elevado e dinámico tráfico de consultas. O éxito e os ingresos recibidos por parte das empresas de Recuperación de Información Web dependen da axilidade de resposta das consultas que reciben e da calidade dos resultados ofrecidos. Para controlar esta situación, as grandes compañías deben construír grandes centros de datos, xeograficamente distribuídos, e compostos por milleiros de servidores. A subministración eléctrica destas colosais infraestruturas supón un altísimo gasto enerxético, e unha pequena mellora a nivel de eficacia pode supor grandes vantaxes económicas.

Esta tese representa unha nova achega ao estado da arte actual referido á xestión de consultas e ao consumo enerxético de grandes centros de datos. Esta investigación permitirá a grandes compañías de Recuperación de Información a construción de motores de busca dotados de maior eficiencia e a súa integración no concepto de *Green Information Retrieval*.

Por unha parte, esta tese propón novas técnicas de distribución de consultas aos servidores que as procesan para diminuír o seu tempo de resposta. Mediante técnicas de predición do tempo de execución das consultas que están en cola, agardando a ser procesadas por un servidor, é posible estimar cal será o que procesara a consulta co menor tempo de espera.

Pola outra parte, esta tese define un modelo matemático sinxelo que establece unha negociación entre o tempo de resposta que ofrece un motor de busca e o consumo eléctrico que xera. Habitualmente, estes parámetros adoitan ser opostos, de xeito que intentar mellorar algún deles supón o empeoramento do outro. As flutuacións do tráfico de consultas ó longo de todo un día son a chave en torno á que xira este modelo: cando o sistema recibe un número de consultas moi elevado, que pode ocasionar maiores tempos de espera, o modelo automaticamente incrementa o número de máquinas activas no sistema e mantén así uns tempos de resposta axeitados. Da mesma forma, se a carga do sistema é baixa, o modelo reduce o número de servidores activos, o que xera grandes porcentaxes de aforro enerxético, especialmente en intervalos de pouca actividade.

Experimentos con diferentes conxuntos de datos, diversas taxas de tráfico e historiais de consultas reais e sintéticos, testemuñan o gran poder, tanto dos métodos de distribución de consultas coma do modelo matemático, para obter grandes melloras en canto a eficiencia e aforro enerxético, con respecto aos métodos de referencia.

B.1 Motivación

A investigación desta tese está motivada por diferentes carencias no campo da Recuperación de Información, para as que esta tese propón diversas solucións.

En primeiro lugar, non existe un amplo estado da arte referente á xestión das consultas de usuario nos motores de busca a grande escala. Xestionar unha consulta fai referencia a enviala ao servidor axeitado para que a procese e xere os resultados correspondentes. As aproximacións xa existentes antes da realización desta tese son verdadeiramente sinxelas, pero asumen condicións inadecuadas, o que xera una xestión ineficiente co correspondente incremento do tempo de resposta das consultas.

A preocupación por reducir o tempo de resposta ofrecido por un motor de busca fai que unha das solución adoptadas sexa ampliar o número de servidores que dan reposta ás consultas dos usuarios. Esta medida, non obstante, incrementa considerablemente o consumo enerxético, o que afecta non ó aos custes económicos das empresas senón tamén ás emisións de CO_2. Actualmente, algunhas empresas de RI están dedicando incontables recursos para diminuír o consumo enerxético: situación estratéxica dos centros de datos para aproveitar enerxías renovables, reutilización de recursos, sistemas máis eficientes... Varias empresas como Google ou Microsoft mesmo publican as súas emisións de carbono e mostran as medidas adoptadas para dar exemplo a outras empresas para que se unan a este comportamento coñecido como *Green IR - Green Information Retrieval*. O control do cambio climático e a sostibilidade pasa por expandir estas costumes a un maior número de empresas do sector tecnolóxico.

Centrándonos nos motores de busca, o tráfico de consultas que reciben varía considerablemente ao longo de todo o día, con períodos de baixa carga ao inicio e final do día, e outros períodos de alto tráfico nas horas centrais. Deste xeito, parte dos servidores que procesan consultas cando o tráfico é elevado, non son utilizados en períodos de baixa actividade, polo que poderían ser apagados, postos en suspensión ou ben dedicados a outras tarefas.

Existen traballos no campo da RI estudando a eficiencia enerxética dos motores de busca. Non obstante, ata onde chega o noso coñecemento, non existen traballos previos encargados de activar ou desactivar servidores dun motor de busca en función da carga de traballo dun centro de datos.

Deste xeito, se en primeiro lugar reducimos o tempo de resposta das consultas, mellorando os algoritmos de xestión das mesas, e en segundo lugar reducimos o consumo enerxético dos motores de busca, poderemos conseguir sistemas máis eficientes co seu correspondente aforro económico, o que constituirá un grande incentivo para as empresas de Recuperación de Información.

B.2 Obxectivos

O principal obxectivo desta tese constitúeo o estudo exhaustivo que permita corroborar os dous enunciados ou hipóteses seguintes:

- A eficiencia dun sistema de Recuperación de Información a grande escala pode ser mellorada usando preditores do tempo de execución das consultas á hora de distribuílas aos servidores axeitados para xerar a súa resposta.

- O consumo enerxético dun motor de busca a grande escala pode reducirse sen comprometer para elo a eficiencia do sistema.

Ámbolos dous obxectivos están dirixidos pola eficiencia do motor de busca. O primeiro persegue directamente diminuír o tempo de resposta que o usuario debe agardar para que a súa consulta sexa resolta. O segundo punto de interese persegue diminuír o consumo enerxético dun motor de busca, mantendo sempre presentes os tempos de resposta das consultas, e reaccionando en base a eles.

B.3 Estrutura

As principais contribucións desta tese preséntanse nos Capítulos 3, 4, 5 e 6. O Capítulo 2 introduce algúns conceptos de RI para os lectores non expertos no campo. A organización dos capítulos da tese é como segue:

- O Capítulo 2 presenta modelos básicos de RI sobre os que se fundamenta esta tese. En particular, defínense conceptos que compoñen o proceso xeral de Recuperación de Información, como son o proceso de indexación e *crawling*. Do mesmo xeito, indicamos como os sistemas de RI evolucionaron grazas ao uso tan estendido da Web, dando lugar aos sistemas de RI web. Estes sistemas obrigaron a construción de grandes sistemas distribuídos, ante a imposibilidade de almacenar toda a ampla colección de material on-line nun único servidor. Estas grandes plataformas dan lugar á definición do termo *Green IR*, tratado nunha das seccións deste capítulo. O concepto de evaluación de sistemas de RI pon fin a este capítulo, dando paso ao seguinte (Capítulo 3) onde se tratará máis en detalle.

- O Capítulo 3 constitúe un estudo das principais plataformas de evaluación de sistemas de Recuperación de Información: sistemas reais, virtualización ou simulación, entre outras. Neste capítulo realízase unha comparanza considerando non só custos financeiros senón tamén a dificultade do desenvolvemento destas aproximacións e mesmo a fiabilidade dos resultados. Desenvolvemos ademais unha parte experimental co obxectivo de comparar a idoneidade das plataformas de evaluación á hora de representar un sistema de RI. Este capítulo permítenos tomar a decisión sobre a plataforma empregada para realizar os experimentos que se detallan nos seguintes capítulos.

- O Capítulo 4 propón un novo método, chamado *Least Loaded*, que mellora as técnicas de xestión de consultas presentes no estado da arte. Esta técnica está baseada en pedir o tempo que unha consulta estará agardando en cada un dos servidores dispoñibles, e dirixila a aquel que ofreza un menor tempo de espera. Para elo empregamos uns predictores que demostran ser uns estimadores moi potentes para o noso caso de estudo. Preséntanse resultados experimentais, usando dous conxuntos de datos, co obxectivo de analizar o funcionamento do método proposto baixo diferentes condicións como poden ser a arquitectura do motor de busca (número de servidores e réplicas) e o tráfico de consultas.

- O Capítulo 5 analiza as desvantaxes do método proposto previamente e combáteas, propoñendo un método híbrido de xestión de consultas, que adopta o comportamento de métodos máis sinxelos cando a carga do sistema é o suficientemente lixeira como para necesitar un método tan potente como *Least Loaded*, e imita a este último en situacións nas que os servidores se saturarían cos métodos tradicionais, e os tempos de resposta non serían aceptables. Esta aproximación permite mellorar o comportamento das técnicas anteriores e, con elo, o estado da arte.

- O Capítulo 6 está motivado principalmente polo concepto de *Green IR*, que se refire á sostibilidade enerxética en grandes centros de Recuperación de Información. Neste capítulo definimos un modelo matemático que establece un balance entre a latencia obtida polo sistema e o consumo enerxético que xera, co obxectivo de desactivar servidores cando a carga do sistema permita que estean libres de traballo. Deste xeito, o modelo automaticamente acende ou pon en suspensión aos servidores, dependendo do tráfico entrante. Experimentos con fluxos de consultas reais permiten comprobar o grande aforro enerxético con respecto aos modelos de referencia e como o noso modelo mantén as porcentaxes de latenza.

- O Capítulo 7 realiza o peche desta tese enumerando as conclusións derivadas deste traballo, así como posibles liñas de traballo que poden ser acometidas nun futuro.

B.4 Conclusións

A continuación expoñemos as principais conclusións obtidas neste traballo de investigación.

B.4.1 Os modelos de simulación son plataformas fiables para desenvolver experimentos a grande escala de Recuperación de Información, conseguindo un grande aforro enerxético.

O Capítulo 3 constitúe un estudo das principais plataformas de evaluación empregadas no campo da Recuperación de Información á hora de estudar a eficiencia dos motores de busca. A elección da plataforma axeitada depende de diferentes factores que estudamos, como a complexidade de representación do modelo real nun modelo analítico ou simulado, as dimensións da arquitectura elixida ou ben dos recursos físicos e económicos que teñamos á nosa disposición. Tras estudar en profundidade todas as aproximacións, concluímos que as plataformas de simulación constitúen una boa alternativa para representar sistemas de RI, xa que permiten representar con alta fiabilidade un sistema real. Por ende, ao permitir aforro de recursos, co seu correspondente aforro enerxético, contribúe aos principios incluídos no concepto de *Green IR*. Este estudo impulsou o uso de simulación como a nosa plataforma de experimentación.

B.4.2 A predicción do tempo de execución das consultas mellora a eficiencia dos motores de busca.

Poucos son os traballos encargados de estudar como enviar as consultas aos servidores que as procesan. *Round Robin* ou *First-In-First-Out* son as aproximacións máis empregadas. Non obstante, o Capítulo 4 explica como estes métodos consideran que todas as consultas teñen o mesmo tempo de resposta. Tras explicar as amplamente usadas técnicas de *pruning* ou poda, foi doado demostrar como estas suposicións non eran correctas. Propoñemos un novo método, chamado *Least Loaded*, que predí o tempo de execución das consultas (*query efficiency predictors*) para calcular o tempo de procesamento de todas as consultas que están na cola esperando a ser procesadas. Deste xeito, podemos estimar o tempo que unha consulta debe esperar en cada un dos servidores de consultas antes de ser procesada. Así, seleccionarase o servidor que ofreza un menor tempo de espera. Para comprobar esta aproximación utilizamos conxuntos de datos tanto sintéticos como reais, con tráfico variable e diferentes arquitecturas distribuídas con distinto número de servidores e réplicas. Con estes experimentos puidemos demostrar que predir a eficiencia das consultas permítenos reducir nunha alta porcentaxe o tempo de espera das consultas, o que supón mellorar a eficiencia dos grandes motores de busca.

B.4.3 A combinación de métodos de xestión de consultas evita a pequena sobrecarga das técnicas de predición e mellora as aproximacións existentes.

O capítulo 5 identifica a desvantaxe de usar preditores de eficiencia das consultas: o tempo empregado no cálculo de ditas predicións supón un pequeno retraso no sistema, que pode aumentar os tempos de espera cando o traico de consultas pode ser servido por métodos máis sinxelos como *Queue Length* o *Round Robin*. Por elo, propomos un método de xestión de consultas híbrido, que aproveita as vantaxes da predición de consultas cando a carga do sistema é alta e combínaas coa simplicidade de outros métodos como *Round Robin* cando o tráfico é baixo. Os experimentos, realizados sobre a colección ClueWeb categoría B de 50 millóns de documentos e usando tráfico de consultas variable e diferentes arquitecturas do motor de busca, demostraron a potencia desta aproximación híbrida para mellorar a eficiencia do sistema.

B.4.4 O modelo matemático proposto consegue obter un balance entre o consumo enerxético e a latencia dun motor de busca, proporcionando un alto aforro enerxético.

O modelo matemático presentado no Capítulo 6 é a principal contribución desta tese. Propoñemos un modelo autoadaptativo para sistemas de busca a grande escala, que permite establecer un balance entre a latencia e o consumo enerxético xerado. Para elo, segundo a carga de consultas vai variando ó longo de todo o día, tamén o fai, dun xeito proporcional e automático, o número de servidores requiridos para responder a esas consultas. Ao inicio e ao final do día, o sistema reduce automaticamente o número de máquinas activas co obxectivo de aforrar enerxía eléctrica. Pola contra, nun escenario de alta carga, como pode ser nas horas centrais do día, o número de máquinas activas pode chegar a alcanzar o seu máximo. O modelo estima o número de máquinas a usar en función da carga de consultas en días anteriores o mesmo considerando o tráfico actual. O modelo acepta diferentes fórmulas para o cálculo da latencia do sistema, parámetro que se ten en conta á hora de activar ou desactivar os servidores. Os experimentos demostraron como o modelo permite alcanzar un aforro enerxético dun 33 % sin degradar a latenza do sistema. Este modelo respecta ademais os principios da *Green IR*, que pouco a pouco está a ser introducida nas empresas de Recuperación de Información.

B.4.5 O modelo $M/M/s$ de Teoría de Colas non é axeitado para estimar a latenza dun motor de busca a grande escala.

O Capítulo 6, e máis especificamente, a Sección 6.7.1, aplica o modelo $M/M/s$ de Teoría de Colas para estimar o tempo que una consulta debe esperar na cola dun servidor para ser procesada. O obxectivo era comprobar la adecuación deste modelo de Teoría de Colas para estimar o valor da latencia dentro do modelo matemático do Capítulo 6. Os experimentos demostraron que, cando a carga do sistema é baixa, permite alcanzar un funcionamento similar ao dos métodos de referencia. Non obstante, ante gran afluencia de tráfico de consultas, o sistema torna non estacionario, e o modelo $M/M/s$ é incapaz de computar unha solución. Nestes casos, a solución adoptada foi utilizar o máximo número de máquinas dispoñibles no sistema, pero esta aproximación non ofrece aforro enerxético con respecto ao método de referencia *Threshold*. Alternativamente, propúxose unha solución determinista que soluciona esta limitación do modelo $M/M/s$ para motores de busca.

B.5 Futuras liñas de investigación

Esta sección expón posibles liñas de traballo que esta tese deixa abertas para futuros desenvolvementos. Estruturamos estas liñas en tres seccións: Plataformas de Evaluación de RI, Xestión de Consultas e Modelo de Balance Latencia/Consumo Enerxético.

B.5.1 Plataformas de Evaluación de RI.

Os experimentos levados a cabo no Capítulo 3 con respecto ás plataformas de virtualización, realizáronse sobre KVM, una plataforma de virtualización amplamente utilizada, co obxectivo de estudar a fiabilidade das plataformas de virtualización para representar sistemas reais de RI. Non obstante, consideramos que sería interesante probar o comportamento doutras plataformas de virtualización que permitan maior flexibilidade á hora de configuralas, para así poder axustar algúns parámetros que a plataforma usada no permitía configurar. Deste xeito, poderíamos encontrar unha plataforma de virtualización que representase máis fielmente un escenario real.

B.5.2 Xestión de Consultas.

Co obxectivo de facer más realistas os métodos de xestión de consultas propostos nesta tese, sería interesante considerar o uso dunha caché de consultas. Nos nosos experimentos, consideramos que una consulta sempre se enviaba a un servidor para ser procesada. Sen embargo, si fixésemos uso dunha caché, permitiría responder algunhas consultas sen necesidade de envialas aos servidores, usando os resultados almacenados de consultas iguais que se responderon previamente.

B.5.3 Modelo de Balance Latenza/Consumo Enerxético.

No escenario representado no Capítulo 6, considerouse que os servidores eran activados ou pasados a modo suspensión, dependendo da tasa de tráfico entrante. O tempo que un servidor ocupa en activarse se está en suspensión, é desprezado. Unha importante liña de investigación sería considerar un escenario máis complexo no que os servidores fosen totalmente apagados cando non fose necesario o seu uso. Este caso requiriría considerar o tempo que un servidor tarda en ser acendido, o que pode xerar retardos importantes que habería que xestionar, pero que permitirá un maior aforro enerxético.

Un posible estudo que completaría o actual sería engadir custos financeiros ó modelo, tal e como figura noutros traballos previos ([86] e [53]). As funcións de custo poderían definirse dun modo máis complexo para considerar varias propiedades importantes como variacións de custo espacio-temporais.

Incluír parámetros hardware no modelo faríao mesmo máis realista. Variables coma o uso de memoria ou de CPU son factores interesantes a considerar para activar ou desactivar un servidor.

B.6 Publicacións

A maior parte do contido desta tese foi xa publicado, deixando patente a contribución deste traballo ó ámbito científico da Recuperación de Información.

- Os contidos desenvolvidos no Capítulo 3 sobre o estudo comparativo de diferentes plataformas de evaluación constitúe a seguinte publicación: *Analysis of performance evaluation techniques for Large Scale Information Retrieval. Ana Freire, Fidel Cacheda, Vreixo Formoso and Víctor Carneiro. In Proceedings of LSDS-IR 2013*.

- O método *Least Loaded* de xestión de consultas presentado no Capítulo 4 foi publicado nun dos máis relevantes congresos de Recuperación de Información (A*): *Scheduling Queries Across Replicas. Ana Freire, Craig Macdonald, Nicola Tonellotto, Iadh Ounis and Fidel Cacheda. In Proceedings of SIGIR 2012*. (36.5 % aceptación).

- O método híbrido de xestión de consultas estudado ó longo do Capítulo 5 deu lugar á seguinte publicación: *Hybrid query scheduling for a replicated search engine. Ana Freire, Craig Macdonald, Nicola Tonellotto, Iadh Ounis and Fidel Cacheda. In Proceedings of ECIR 2013*. (29 % aceptación).

- Os contidos que engloba o Capítulo 6 foron recentemente publicados: *A Self-Adapting Latency/Power Tradeoff Model for Replicated Search Engines. Ana Freire, Craig Macdonald, Nicola Tonellotto, Iadh Ounis and Fidel Cacheda. In Proceedings of WSDM 2014*. (18 % aceptación).

Bibliography

[1] Green it: The new industry shockwave, April 2007. Gartner Inc.

[2] Cloudorado. `http://www.cloudorado.com/`, 2014. [Online; accessed January-2014].

[3] Findthebest. `http://cloud-computing.findthebest.com/`, 2014. [Online; accessed January-2014].

[4] G. Amati, E. Ambrosi, M. Bianchi, C. Gaibisso, and G. Gambosi. FUB, IASI-CNR and University of Tor Vergata at TREC 2007 Blog track. In *Proc. of TREC*, 2007.

[5] V. N. Anh and A. Moffat. Pruned query evaluation using pre-computed impacts. In *Proceedings of SIGIR 2006*, pages 372–379, 2006.

[6] D. Arroyuelo, V. G. Costa, S. González, M. Marín, and M. Oyarzún. Distributed search based on self-indexed compressed text. *Inf. Process. Manage.*, 48(5):819–827, 2012.

[7] C. Badue, R. Baeza-yates, B. Ribeiro-neto, and N. Ziviani. Distributed query processing using partitioned inverted files. In *In Proc. of the 9th String Processing and Information Retrieval Symposium (SPIRE*, pages 10–20. IEEE CS Press, 2001.

[8] C. S. Badue, R. A. Baeza-Yates, B. A. Ribeiro-Neto, A. Ziviani, and N. Ziviani. Analyzing imbalance among homogeneous index servers in a web search system. *Inf. Process. Manage.*, 43(3):592–608, 2007.

[9] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, May 1999.

[10] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology behind Search (2nd Edition) (ACM Press Books)*. Addison-Wesley Professional, 2 edition, February 2011.

[11] L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, mar 2003.

[12] T. Berners-Lee, R. Cailliau, and J.-F. Groff. The world-wide web. *Computer Networks and ISDN Systems*, 25(4-5):454–459, 1992.

[13] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition, 2000.

[14] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc. WWW*, pages 107–117, 1998.

[15] D. Broccolo, C. Macdonald, S. Orlando, I. Ounis, R. Perego, F. Silvestri, and N. Tonellotto. Load-sensitive selective pruning for distributed search. In *Proceedings of the 22Nd ACM International Conference on Conference on Information &#38; Knowledge Management*, CIKM '13, pages 379–388, New York, NY, USA, 2013.

[16] D. Broccolo, C. Macdonald, S. Orlando, I. Ounis, R. Perego, F. Silvestri, and N. Tonellotto. Query processing in highly-loaded search engines. In O. Kurland, M. Lewenstein, and E. Porat, editors, *SPIRE*, volume 8214 of *Lecture Notes in Computer Science*, pages 49–55. Springer, 2013.

[17] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the twelfth international conference on Information and knowledge management*, CIKM '03, pages 426–434, New York, NY, USA, 2003.

[18] J. D. Brutlag, H. Hutchinson, and M. Stone. User preference and search engine latency. In *JSM Proceedings, Qualtiy and Productivity Research Section*, 2008.

[19] S. Buettcher, C. L. A. Clarke, and G. V. Cormack. *Information Retrieval: Implementing and Evaluating Search Engines*. The MIT Press, 1st edition, 2010.

[20] D. Bunday. *An Introduction to Queueing Theory*. Arnold, 1996.

[21] F. J. Burkowski. Retrieval performance of a distributed text database utilizing a parallel processor document server. In *Proceedings of the second international symposium on Databases in parallel and distributed systems*, DPDS '90, pages 71–79, New York, NY, USA, 1990.

[22] F. Cacheda, V. Carneiro, D. Fernandez, and V. Formoso. Performance evaluation of large-scale information retrieval systems scaling down. In *Workshop on Large-Scale Distributed Information Retrieval*, 2010.

[23] F. Cacheda, V. Carneiro, V. Plachouras, and I. Ounis. Performance analysis of distributed information retrieval architectures using an improved network simulation model. *Inf. Process. Manage.*, 43(1):204–224, jan 2007.

[24] F. Cacheda, V. Carneiro, V. Plachouras, and I. Ounis. Performance comparison of clustered and replicated information retrieval systems. In *Proceedings of the 29th European conference on IR research*, ECIR'07, pages 124–135, Berlin, Heidelberg, 2007.

[25] F. Cacheda, J. M. F. Luna, and J. F. H. Guadix. *Recuperación de información: un enfoque práctico y multidisciplinar*. RA-MA editorial y Publicaciones. S.A., 2011.

[26] F. Cacheda, V. Plachouras, and I. Ounis. A case study of distributed information retrieval architectures to index one terabyte of text. *Inf. Process. Manage.*, 41(5):1141–1161, 2005.

[27] B. Cahoon and K. McKinley. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Trans. on Information Systems*, 18:1–43, 1997.

[28] B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. Early exit optimizations for additive machine learned ranking systems. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 411–420, New York, NY, USA, 2010.

[29] R. Cao. *Introducción a al Simulación y a la Teoría de Colas*. Carlos Iglesias, España, 1st edition, 2002.

[30] A. L. Chervenak. Performance measurements of the first raid prototype. Technical report, Department of Computer Science, University of California, Berkeley, 1990.

[31] A. Chowdhury and G. Pass. Operational requirements for scalable search systems. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 435–442, New York, NY, USA, 2003.

[32] G. Chowdhury. An agenda for green information retrieval research. *Inf. Process. Manage.*, 48(6):1067–1077, 2012.

[33] R. B. Cooper. Queueing theory. In *Encyclopedia of Computer Science*, pages 1496–1498. John Wiley and Sons Ltd., Chichester, UK, 2000.

[34] N. Craswell, R. Jones, G. Dupret, and E.Viegas, editors. *Proceedings of the Web Search Click Data Workshop at WSDM 2009*, 2009.

[35] J. Dean. Challenges in building large-scale information retrieval systems: invited talk. In *Proceedings of the Second International Conference on Web Search and Web Data Mining, WSDM 2009*, New York, NY, USA, 2009.

[36] Y. Diao, N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury. Using mimo feedback control to enforce policies for interrelated metrics with application to the apache web server. In *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP*, pages 219–234, 2002.

[37] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[38] D. Economou, S. Rivoire, and C. Kozyrakis. Full-system power analysis and modeling for server environments. In *In Workshop on Modeling Benchmarking and Simulation (MOBS)*, 2006.

[39] M. Fontoura, V. Josifovski, J. Liu, S. Venkatesan, X. Zhu, and J. Zien. Evaluation strategies for top-k queries over memory-resident inverted indexes. *Proceedings of the VLDB Endowment*, 4(12):1213–1224, August 2011.

[40] M. G. for Java. http://mg4j.dsi.unimi.it, 2014. [Online; accessed January-2014].

[41] A. Freire, F. Cacheda, V. Formoso, and V. Carneiro. Analysis of performance evaluation techniques for large scale information retrieval. 2013.

[42] A. Freire, C. Macdonald, N. Tonellotto, I. Ounis, and F. Cacheda. Scheduling queries across replicas. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 1139–1140, New York, NY, USA, 2012.

[43] A. Freire, C. Macdonald, N. Tonellotto, I. Ounis, and F. Cacheda. Hybrid query scheduling for a replicated search engine. In *Advances in Information Retrieval*, volume 7814, pages 435–446, 2013.

[44] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[45] A. Gandhi and M. Harchol-Balter. How data center size impacts the effectiveness of dynamic power management. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 1164–1169, 2011.

[46] Y. Ganjisaffar, R. Caruana, and C. Lopes. Bagging gradient-boosted trees for high precision, low variance ranking models. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information*, SIGIR '11, pages 85–94, New York, NY, USA, 2011.

[47] B. He and I. Ounis. Inferring query performance using pre-retrieval predictors. In *In Proc. Symposium on String Processing and Information Retrieval*, pages 43–54. Springer Verlag, 2004.

[48] B. He and I. Ounis. Query performance prediction. *Inf. Syst.*, 31(7):585–594, Nov. 2006.

[49] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[50] U. Hoelzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[51] B. Jeong and E. Omiecinski. Inverted file partitioning schemes in multiple disk systems. *IEEE Transactions on Parallel and Distributed Systems*, 6:142–153, 1995.

[52] S. Jonassen, B. B. Cambazoglu, and F. Silvestri. Prefetching query results and its impact on search engines. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '12, pages 631–640, New York, NY, USA, 2012.

[53] E. Kayaaslan, B. B. Cambazoglu, R. Blanco, F. P. Junqueira, and C. Aykanat. Energy-price-driven query processing in multi-center web search engines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pages 983–992, New York, NY, USA, 2011.

[54] D. G. Kendall. Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. *Annals of Mathematical Statistics*, 24(3):338–354, 1953.

[55] B. Khargharia, S. Hariri, and M. S. Yousif. Autonomic power and performance management for computing systems. *Cluster Computing*, 11(2):167–181, June 2008.

[56] E. Kharitonov, C. Macdonald, P. Serdyukov, and I. Ounis. Incorporating efficiency in evaluation. In *Proceedings of the SIGIR 2013 Workshop on Modeling User Behavior for Information Retrieval Evaluation (MUBE 2013)*, August 2013.

[57] KVM. http://www.linux-kvm.org, 2014. [Online; accessed January-2014].

[58] J. Leung. *Handbook of Scheduling*. Chapman & Hall, 2004.

[59] J. Liu, F. Zhao, X. Liu, and W. He. Challenges towards elastic power management in internet data centers. *2012 32nd International Conference on Distributed Computing Systems Workshops*, 0:65–72, 2009.

[60] M. Livny. DeNet user's guide. Technical report, University of Wisconsin, Madison, 1990.

[61] J. Lu and J. Callan. Content-based retrieval in hybrid peer-to-peer networks. In *Proceedings of CIKM 2003*, pages 199–206, 2003.

[62] Z. Lu and K. S. McKinley. Partial collection replication versus caching for information retrieval systems. In *ACM International Conference on research and development in information retrieval*, pages 248–255, 2000.

[63] R. López and F. Cacheda. *A Technical Approach to Information Retrieval Pedagogy*, pages 89–105. Springer, 2011.

[64] C. Macdonald. *The Voting Model for People Search*. PhD thesis, University of Glasgow, February 2009.

[65] C. Macdonald, R. McCreadie, R. Santos, and I. Ounis. From puppy to maturity: Experiences in developing terrier. In *Proc. of the OSIR at SIGIR 2012*, 2012.

[66] C. Macdonald, I. Ounis, and N. Tonellotto. Upper bound approximations for dynamic pruning. *Transactions on Information Systems*, 29(4), 2011.

[67] C. Macdonald, N. Tonellotto, and I. Ounis. Learning to predict response times for online query scheduling. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '12, pages 621–630, New York, NY, USA, 2012.

[68] A. Macfarlane, J. A. Mccann, and S. E. Robertson. Parallel search using partitioned inverted files. In *In 7th International Symposium on String Processing and Information Retrieval*, pages 209–220. IEEE Computer Society, 2000.

[69] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[70] M. Marin and V. Gil-Costa. High-performance distributed inverted files. In *Proceedings of the 16th ACM conference on Conference on information and knowledge management*, CIKM '07, pages 935–938, New York, NY, USA, 2007.

[71] M. Marin, V. Gil-Costa, and C. Gomez-Pantoja. New caching techniques for web search engines. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 215–226, New York, NY, USA, 2010.

[72] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi. Cloud computing - the business perspective. *Decis. Support Syst.*, 51(1):176–189, Apr. 2011.

[73] M. Marzolla and L. C. libcppsim: A simula-like, portable process-oriented simulation library in c++, 2004.

[74] L. Mastroleon, N. Bambos, C. Kozyrakis, and D. Economou. Automatic power management schemes for internet servers and data centers. In *GLOBECOM*, page 5. IEEE, 2005.

[75] A. Moffat, W. Webber, and J. Zobel. Load balancing for term-distributed parallel retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in IR*, SIGIR '06, pages 348–355, New York, NY, USA, 2006.

[76] A. Moffat, W. Webber, J. Zobel, and R. Baeza-Yates. A pipelined architecture for distributed text query evaluation. *Inf. Retr.*, 10:205–231, June 2007.

[77] A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *Transactions on Information Systems*, 14(4):349–379, 1996.

[78] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. Terrier: A high performance and scalable information retrieval platform. In *Proceedings of the OSIR Workshop 2006*, pages 18–25, France, August 2006.

[79] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Real-Time Syst.*, 23(1/2):127–141, 2002.

[80] M. Persin. Document filtering for fast ranking. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, pages 339–348, New York, NY, USA, 1994.

[81] R. R. Pollán and Álvaro Barreiro. Enabling the grid for experiments in distributed information retrieval. In B. M. et al., editor, *Proceedings of the First EELA-2 Conference*, 2008.

[82] K. Radinsky, K. Svore, S. Dumais, J. Teevan, A. Bocharov, and E. Horvitz. Modeling and predicting behavioral dynamics on the web. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 599–608, New York, NY, USA, 2012.

[83] B. Ribeiro-Neto and R. Barbosa. *Query performance for tightly coupled distributed digital libraries*, pages 182–190. 1998.

[84] C. J. V. Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.

[85] S. Robertson, S. Walker, M. Hancock-Beaulieu, A. Gull, and M. Lau. Okapi at trec. In *In Proceedings of TREC 1*, pages 21–30, 1992.

[86] F. B. Sazoglu, B. B. Cambazoglu, R. Ozcan, I. S. Altingovde, and O. Ulusoy. A financial cost metric for result caching. In *Proc. of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 873–876, New York, NY, USA, 2013.

[87] E. Shurman and J. Brutlag. Performance related changes and their user impacts. In *Velocity: Web Performance and Operations Conference*, 2009.

[88] F. Silvestri. Mining query logs: Turning search usage data into knowledge. *Foundations and Trends in Information Retrieval*, 4(1-2):1–174, 2010.

[89] B. Simmons, A. McCloskey, and H. Lutfiyya. Dynamic provisioning of resources in data centers. In *Proceedings of ICAS 2007*, pages 40–46, 2007.

[90] E. E. Star. Energy calculator for pc equipment. http://eu-energystar.org/en/en_008.shtml, 2014. [Online; accessed January-2014].

[91] A. Tomasic and H. Garcia-molina. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In *In Proceed-*

ings of the Second International Conference on Parallel and Distributed Information Systems, pages 8–17, 1993.

[92] N. Tonellotto, C. Macdonald, and I. Ounis. Query efficiency prediction for dynamic pruning. In *Proceedings of the 9th workshop on Large-scale and distributed informational retrieval*, LSDS-IR '11, pages 3–8, New York, NY, USA, 2011.

[93] H. Turtle and J. Flood. Query evaluation: strategies and optimizations. *Inf. Process. Manage.*, 31(6):831–850, 1995.

[94] L. Wang, J. Lin, and D. Metzler. Learning to efficiently rank. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 138–145, New York, NY, USA, 2010.

[95] Z. Wang, N. Tolia, and C. Bash. Opportunities and challenges to unify workload, power, and cooling management in data centers. *SIGOPS Oper. Syst. Rev.*, 44(3):41–46, Aug. 2010.

[96] W. Webber. *Design and Evaluation of a Pipelined Distributed Information Retrieval Architecture*. University of Melbourne, Department of Computer Science and Software Engineering, 2007.

[97] Y. Zhao, F. Scholer, and Y. Tsegay. Effective pre-retrieval query performance prediction using similarity and variability evidence, 2008.