A Self-Adapting Latency/Power Tradeoff Model for Replicated Search Engines

Ana Freire[1], Craig Macdonald[2], Nicola Tonellotto[3], Iadh Ounis[2], Fidel Cacheda[1]

[1] University of A Coruña, Campus de Elviña s/n, 15017 A Coruña, Spain
[2] University of Glasgow, G12 8QQ Glasgow, UK
[3] National Research Council of Italy, Via G. Moruzzi 1, 56124 Pisa, Italy

{ana.freire, fidel.cacheda}@udc.es[1], {craig.macdonald, iadh.ounis}@glasgow.ac.uk[2], {nicola.tonellotto}@isti.cnr.it[3]

ABSTRACT

For many search settings, distributed/replicated search engines deploy a large number of machines to ensure efficient retrieval. This paper investigates how the power consumption of a replicated search engine can be automatically reduced when the system has low contention, without compromising its efficiency. We propose a novel self-adapting model to analyse the trade-off between latency and power consumption for distributed search engines. When query volumes are high and there is contention for the resources, the model automatically increases the necessary number of active machines in the system to maintain acceptable query response times. On the other hand, when the load of the system is low and the queries can be served easily, the model is able to reduce the number of active machines, leading to power savings. The model bases its decisions on examining the current and historical query loads of the search engine. Our proposal is formulated as a general dynamic decision problem, which can be quickly solved by dynamic programming in response to changing query loads. Thorough experiments are conducted to validate the usefulness of the proposed adaptive model using historical Web search traffic submitted to a commercial 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 10 ms compared to a baseline that provisions replicas based on a previous day's traffic.

Categories & Subject Descriptors: H.3.3 [Information Storage & Retrieval]: Information Search & Retrieval

Keywords: Search Engines; Power Consumption.

1. INTRODUCTION

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, electricity costs (including power and cooling) form an important part of the operational costs of search engine companies [14]. Indeed, in 2011, Google's overall power consumption was reported[1] to be 2.68 million MWh.

The user query volume typically received by a Web search engine varies through the course of the day [27]. 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 [13]. 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. Instead, 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 argue that a tradeoff can be enforced between the machines devoted to processing queries and query processing deadlines, and that this tradeoff 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 tradeoff can be adapted dynamically to changing query volumes in very short times. To attain this, 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 [2]. Moreover, we provide thorough ex-

WSDM'14, February 24–28, 2014, New York, New York, USA.
Copyright 2014 ACM 978-1-4503-2351-2/14/02 ...$15.00.
http://dx.doi.org/10.1145/2556195.2556246.

[1]Google Green: `http://www.google.com/green/`.

periments 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.

The main contributions of this paper are the following:

- We propose a self-adapting model for replicated 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.
- We show 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 demonstrate 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.

This paper is structured as follows: Section 2 discusses the existing literature on distributed search and green efforts in information technology. In Section 3, we introduce our model, by describing the dynamic system and the general cost function, while Section 4 presents the deterministic approach that considers the previous and also the current state of the system for predicting query traffic. Section 5 proposes power and latency cost functions. In Section 6, we concretely state the research questions that we investigate, as well as detailing the baselines and experimental setup. Section 7 reports our experimental results, with concluding remarks following in Section 8.

2. BACKGROUND

As well as effectively answering the users' information needs, a search engine must be efficient, as users are not willing to wait long for queries to be answered [26]. For large settings, the efficient answering of queries typically encompasses the distribution of the search engine index across multiple machines. As our work is concerned with the analysis of distributed search engines, and their power consumption, in the remainder of this section we provide the necessary background on distributed search engine architectures (Section 2.1) and on previous work on reducing power consumption within information technology (IT) (Section 2.2).

2.1 Distributed/Replicated Search

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. When a query arrives at a *query broker*, it broadcasts the query to all shards, in order to later collect and merge the results and produce the final top K retrieved set for presentation to the user [1]. To ensure high throughput rates, shards are often replicated, so that one of multiple *query servers* can provide the results for a single shard [4]. 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, as well as providing fault tolerance properties. Without loss, we focus on a single broker, single-shard environment with multiple replicas on different query

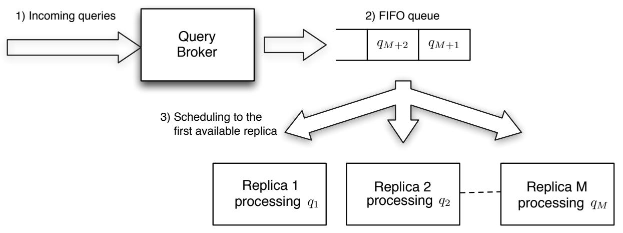
1) Incoming queries
Query Broker
2) FIFO queue
q_{M+2}
q_{M+1}
3) Scheduling to the first available replica
Replica 1 processing q_1
Replica 2 processing q_2
Replica M processing q_M

Figure 1: Our reference architecture.

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 paper could easily be applied for multiple shard environments, by independent application on each shard individually.

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 (Fig. 1). Queries are received by the broker and are queued up in a buffer. While complex queue re-ordering strategies can benefit overall response times [9, 20], 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.

2.2 Green IT

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 [11]. Much research effort has been dedicated to achieve power savings for data centres or Internet servers [8, 17, 19, 21, 30] taking into account different factors such as the devices' power consumption and/or cooling systems. Moreover, large IT companies such as Microsoft[2] and Google[3] 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 [19], where Lin et al. showed that the most effective and aggressive 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.

Many of the afore-mentioned works are focused on achieving power savings in data centres for general Internet services (e.g. [19, 21]). However, few works have addressed this problem within search engines, where user queries equate to (short-lived) jobs. Chowdhury [5] recently introduced the term *Green IR*, that maps this concern into the information retrieval field. His roadmap for improving environmental impact of information retrieval systems addresses the power efficiency of end-user devices as well as within the search engine itself. Within his roadmap, our work is clearly focused on the intra-data centre efficiency of the search engine.

[2]`http://www.microsoft.com/environment/`
[3]`http://www.google.com/green/`

In contrast, the work of [16] 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. [25] 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-data centre model that turns the servers on or off depending on the incoming query traffic needs. Indeed, our model, proposed in the next section, 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.

3. DYNAMIC OPTIMISATION MODEL

We consider a replicated search engine processing user queries, as explained above with reference to Figure 1. 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 from 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 thus *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 required 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 [2]. Computing systems have been previously modelled as dynamic systems in order to leverage automatic control theory to address the dynamics of resource management [12], such as email server [23] and web servers [6]. 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 [2] (Section 3.1), a dynamic model of a replicated search engine with multiple query processors (Section 3.2), a discussion on the cost function for our dynamic model (Section 3.3), and a summary of the resulting decision problem (Section 3.4).

3.1 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* [2]. 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 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:

Table 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

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

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 (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:

$$J(x_0) = E\left[g_N(x_N) + \sum_{k=0}^{N-1} g_k\big(x_k, u_k, w_k\big)\right] \tag{2}$$

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 $J(x_0)$.

3.2 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 will immediately start to process another query. Thus, the number of queries waiting to be processed evolves according to the following discrete-time equation:

$$x_{k+1} = x_k - y_k + w_k \tag{3}$$

The dynamic equation (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:

$$y_k = h_k(x_k, u_k, v_k) \tag{4}$$

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:

$$x_{k+1} = x_k - h_k(x_k, u_k, v_k) + w_k \tag{5}$$

where we leave to later in Section 4 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.

3.3 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 [26], and may abandon their search request [18] if the search engine takes too long to respond. 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 [27], 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 tradeoff, we propose cost combinations of the following type:

$$g_k(\cdot) = \lambda P_k(\cdot) + (1-\lambda) L_k(\cdot) \tag{6}$$

for various values of $\lambda \in [0, 1)$. For $\lambda = 0$, the cost function represented by Equation (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 Section 5, we devise particular cost functions ranging in the [0,1] interval, where 0 means no cost and 1 means maximum cost.

3.4 Latency/Power Decision Problem

We now formulate the general dynamic decision problem by adapting Equations (1) & (2) to our search engine and cost models:

$$\begin{aligned}
\underset{u_k}{\text{minimise}} \quad & E\Bigg[\sum_{k=0}^{N-1} \lambda P_k(x_k, u_k, v_k, w_k) + \\
& \qquad (1-\lambda) L_k(x_k, u_k, v_k, w_k)\Bigg] \qquad (7) \\
\text{subject to} \quad & x_{k+1} = x_k - h_k(x_k, u_k, v_k) + w_k, \\
& k = 0, 1, \ldots, N-1
\end{aligned}$$

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 (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 (7) to be sub-optimally solved in a deterministic manner.

4. DETERMINISTIC APPROXIMATION

Problems like (7) cannot typically be solved analytically, and their solution algorithms are computationally very intensive [2]. For these reasons, these problems, where the exact value of all variables are unknown deterministically, are solved sub-optimally in practice. In order to deal with the stochastic decision problem (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 4.1);
2. how to derive a deterministic approximation for problem (7) with query processing functions $h_k(\cdot)$ depending on our estimations (Section 4.2);
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 4.3).

4.1 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 [3]. 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., $\bar{v}_k = v_{k-SN}$, where S defines the step, i.e., the number of previous days, and N is the total number of time slots in a day. For instance, $S = 1$ means that $\bar{v}_k$ is estimated using data from the previous day, while $S = 7$ 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.:
$$\bar{w}_k = w_{k-SN}$$
- In the *seasonal estimator with drift*, the previous day value is adjusted with the current trend of arrivals [24] experienced in the last two time slots, such that:
$$\bar{w}_k = w_{k-SN} + (w_{k-1} - w_{k-2})$$

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.

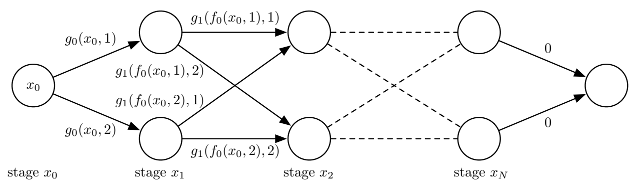

Figure 2: Transition graph for a deterministic problem with 2 machines.

4.2 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:

$$h_k(\cdot) = u_k \cdot T_s/v_k \tag{8}$$

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 (7) with the random variables' estimated values leads to the following deterministic problem formulation:

$$\begin{aligned} \underset{u_k}{\text{minimise}} \quad & \sum_{k=0}^{N-1} \lambda P_k(x_k, u_k, \bar{v}_k, \bar{w}_k) + \\ & (1-\lambda) L_k(x_k, u_k, \bar{v}_k, \bar{w}_k) \\ \text{subject to} \quad & x_{k+1} = \max\{0, x_k - u_k \cdot T_s/\bar{v}_k + \bar{w}_k\} \\ & k = 0, 1, \dots, N-1 \end{aligned} \tag{9}$$

where the $\max\{\cdot\}$ function avoids to process more queries than the number of queries available to process. In contrast to problem (7), problem (9) is deterministic as there is no longer any probability distributions associated to v_k and w_k (and hence x_k). Such deterministic formulation can be tractably solved [2], as described next.

4.3 Deterministic Problem Solutions

Consider the deterministic problem formulation defined in Equation (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 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 (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 [2], but also algorithms specifically designed for the shortest path problem solution, such as the Dijkstra algorithm [7]. 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 [2]. 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[4]. We will denote this solution algorithm with ShortTerm. Moreover, as discussed in Section 4.1, we expect ShortTerm to improve LongTerm, as it considers the query volume being currently experienced by the search engine in addition to the volume experienced on a previous day.

5. COST FUNCTIONS

The previous two sections define our model for balancing power/latency tradeoff, as well as a deterministic approximation. In this section, we study two cost functions to analytically express the power/latency tradeoff as the linear combination expressed in Equation (6). These cost functions allow us to model the benefits/drawback of using a variable number of machines.

5.1 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}.
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 to 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 [10], 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.

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

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 energy 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 energy for M machines, we obtain the following expression for the power cost function $P_k(\cdot)$:

$$P_k(\cdot) = P(u_k) = \frac{1}{MP_{\text{on}}} \left[P_{\text{on}} u_k + P_{\text{standby}} (M - u_k) \right] \quad (10)$$

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.

5.2 Latency Cost Function

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:

$$T_k = \frac{x_k + \bar{w}_k}{u_k} \bar{v}_k \quad (11)$$

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.

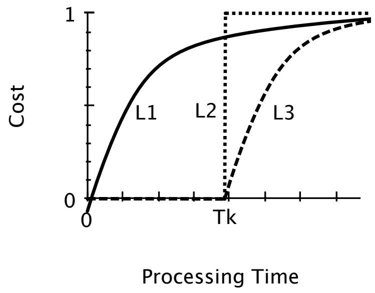

Figure 3: Illustration of latency cost function shapes.

To normalise this completion time in the [0,1] interval, we adapt the latency metrics in [29]:

$$L_k^1(\cdot) = 1 - \exp(\alpha T_k) \tag{12}$$

$$L_k^2(\cdot) = \begin{cases} 0 & \text{if } T_k \leq T_s \\ 1 & \text{otherwise} \end{cases} \tag{13}$$

$$L_k^3(\cdot) = \begin{cases} 0 & \text{if } T_k \leq T_s \\ 1 - \exp(\alpha(T_k - T_s)) & \text{otherwise} \end{cases} \tag{14}$$

While all metrics consider processing of all queries in the time slot, in contrast to L_k^1, 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 (T_s). Figure 3 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.

6. 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, three 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. How do power and efficiency properties of LongTerm and ShortTerm differ?
3. How should the latency cost function be modelled within our self-adaptive models?

In the remainder of this section, we define the experimental setup to address these research questions, covering the search engine (Section 6.1), evaluation measures (Section 6.2), baselines (Section 6.3), and parameter settings (Section 6.4).

6.1 Search Engine, Documents & Queries

To evaluate the proposed model, we determine the processing times for real user queries submitted to a search engine platform. In particular, we index 50M Web documents from the TREC ClueWeb09 corpus (category B) using the Terrier IR platform[5] [22] – 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[6]. In particular, we use queries from 19th May 2006 as testing and - by setting $S = 11$ - use the historical queries from 8th May 2006 as training data for estimating the random variables. Figure 4 presents the number of queries over the course of each day. During retrieval, we use the Wand dynamic pruning technique applying BM25 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, with inverted indexes are stored on a 160GB SATA drive.

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

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

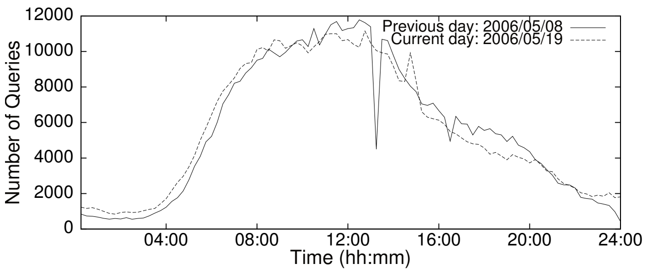

Figure 4: Number of queries arriving per 15 minute slot for both days.

6.2 Evaluation Measures

As our work concerns balancing the tradeoff between search engine efficiency and power consumption, we measure both aspects within our work. In particular, we measure the mean and 90th percentile response time for queries as the configuration of the search engine is varied across the evaluation period (denoted ACT and 90%CT, 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.

6.3 Baselines

To evaluate our proposed model, we define two reasonable baselines for determining u_k for $i = 0, \ldots, N-1$, 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 [13], 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 (11). In doing this, we can derive the decisions u_k for $i = 0, \ldots, N-1$ that 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 (11), 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.

6.4 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 [28] for a small server as follows: $P_{on} = 62W$, $P_{standby} = 2W$, following the reported use of commodity-sized servers within commercial search engines [13]. Within latency cost functions (Equations (12)-(14)), we follow Wang et al. [29] 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 tradeoff λ are experimental variables that we vary within the next section.

7. 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. In particular, Section 7.1 firstly determines the efficiency and power properties of our two baselines. Section 7.2 addresses our first and second research questions, by comparing the LongTerm and ShortTerm approaches with the baselines and with each other. Finally, in Section 7.3, we address our final research question concerning the choice of latency cost function within the model.

7.1 Baselines

The top part of Table 2 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 actives the number of replicas 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 2, 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 expected the Threshold approach results in markedly decreased energy consumption compared to Naïve, with little marked impact on response times.

7.2 Self-adaptive Power/Latency Models

The bottom two parts of Table 2 report the evaluation measures for the LongTerm and ShortTerm approaches. For both approaches and both $M = 15, 20$ replicas, we vary the power/ latency tradeoff parameter λ, to determine its impact on both efficiency and energy consumption. In this section, the L_1 latency cost function (Equation (12)) 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 6.3KWh 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 2. Recall, as explained in Section 4, 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. 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

Table 2: Performance comparison among LongTerm, ShortTerm and the Baselines for different M and λ values.

λ	ACT(ms)	90%CT	% UQ	Max Mch	E(KWh)	ACT(ms)	90%CT	% UQ	Max Mch	E(KWh)
	$M = 15$					$M = 20$				
Baselines										
Naïve	847	1,349	0	15	22.3	846	1,349	0	20	29.8
Threshold	853	1,354	0.03	15	12.0	853	1,354	0.03	19	12.9
LongTerm										
0.25	848	1,350	0	15	12.0	848	1,349	0	17	12.1
0.5	969	1,394	1.35	6	6.3	896	1,409	0.08	7	7.4
0.75	3,194	3,886	19.19	2	2.7	2,299	3,049	4.68	3	4.2
ShortTerm										
0.25	857	1,356	0.03	8	8.0	854	1,354	0.02	10	9.5
0.5	1,536	2,260	1.36	4	4.8	1,007	1,567	0.31	5	5.7
0.75	3,225	3,724	11.84	2	2.9	2,956	3,700	10.69	2	3.4

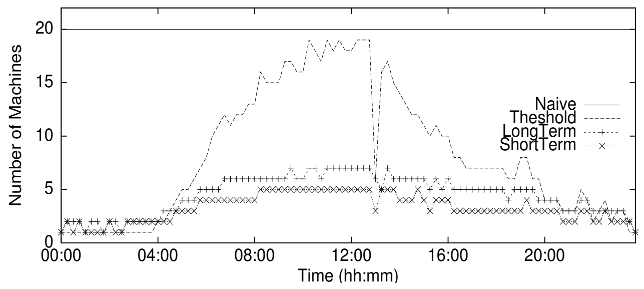

Figure 5: Number of machines used along the day by LongTerm and ShortTerm ($\lambda = 0.5$, $M = 20$) and the two baselines, namely Naïve and Threshold.

savings. This is clearly illustrated in Figure 5, 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 shown in Table 2.

In summary, in answer to our first research question, we find that our proposed self-adapting models can markedly reduce power consumption, without marked impact on efficiency. With respect to the second 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 along the day, 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%.

7.3 Modelling Latency Costs

In this section, we address our final research question, examining how the latency of the search engine should be modelled within our proposed approach. Firstly, recall that Wang et al. [29] proposed three different efficiency metrics, which we used as inspiration for efficiency cost functions in Section 5.2. In particular, the latency of the search engine

Table 3: Comparison of latency functions, while varying λ.

	ACT(ms)	90%CT	% 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

can be modelled with an exponential decay function (Equation (12), denoted L^1), a step function with a fixed penalty after a time threshold has expired (Equation (13), L^2) or a step function followed by an exponential decay (Equations (14), L^3). As we consider the processing of queries within a time slot, we set the time threshold as the slot length $T_s = 15$ minutes.

Table 3 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 3, we observe that latency function L^1 achieves the lowest response times and energy consumptions. This can be explained with reference to Figure 3: For L^2, the cost function 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 final 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 (12).

8. CONCLUSIONS

While various have been dedicated to reduce power consumption regarding IT systems, few works apply this concept to the information retrieval field. In this work we propose 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 (calculates the number of necessary machines to ensure latency values under a threshold), using 1 million queries submitted to a real commercial search engine. Our results show that our self-adapting model can achieve an energy savings 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, while more substantial energy savings can be attained while accepting marginally larger efficiency degradations.

In this paper, 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 [15]. For future work, we will consider a more complex scenario where servers are fully powered down when not required, but incur delay on startup.

9. ACKNOWLEDGEMENTS

Ana Freire and Fidel Cacheda acknowledge the support from the Spanish Government (TIN2009-14203), Xunta de Galicia (CN2012/211) and FEDER funds. Craig Macdonald and Iadh Ounis acknowledge the support of EC-funded project SMART (FP7-287583). Nicola Tonellotto acknowledges the support of EC-funded projects InGeoClouds (FP7-297300) and MIDAS (FP7-318786).

10. REFERENCES

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

[2] D. P. Bertsekas. *Dynamic programming and optimal control.* Athena Scientific, 2nd ed, 2000.

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

[4] 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, 2007.

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

[6] 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 *Proc. of the Network Operations and Management Symposium 2002*, pages 219–234.

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

[8] D. Economou, S. Rivoire, and C. Kozyrakis. Full-system power analysis and modeling for server environments. In *Proc. of the Workshop on Modeling Benchmarking and Simulation*, 2006.

[9] A. Freire, C. Macdonald, N. Tonellotto, I. Ounis, and F. Cacheda. Hybrid query scheduling for a replicated search engine. In *Proc. of ECIR 2013*, pages 435–446.

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

[11] Gartner. Green IT: The new industry shockwave. *Presentation at Symposium/ITXPO conference*, 2007.

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

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

[14] 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 ed, 2009.

[15] S. Jonassen, B. B. Cambazoglu, and F. Silvestri. Prefetching query results and its impact on search engines. In *Proc. of SIGIR 2012*, pages 631–640.

[16] 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 *Proc. of SIGIR 2011*, pages 983–992.

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

[18] E. Kharitonov, C. Macdonald, P. Serdyukov, and I. Ounis. Incorporating efficiency in evaluation. In *Proc. of the Modeling User Behavior for Information Retrieval Evaluation Workshop at SIGIR 2013.*

[19] J. Liu, F. Zhao, X. Liu, and W. He. Challenges towards elastic power management in Internet data centers. *Proc. of Cyber-Physical Systems Workshop at ICDCS 2009.*

[20] C. Macdonald, N. Tonellotto, and I. Ounis. Learning to predict response times for online query scheduling. In *Proc. of SIGIR 2012*, pages 621–630.

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

[22] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. Terrier: A high performance and scalable information retrieval platform. In *Proc. of the Open Source IR Workshop at SIGIR 2006*, pages 18–25.

[23] 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.

[24] K. Radinsky, K. Svore, S. Dumais, J. Teevan, A. Bocharov, and E. Horvitz. Modeling and predicting behavioral dynamics on the web. In *Proc. WWW 2012*, pages 599–608.

[25] F. B. Sazoglu, B. B. Cambazoglu, R. Ozcan, I. S. Altingovde, and O. Ulusoy A financial cost metric for result caching. In *Proc. of SIGIR 2013*, pages 873–876.

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

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

[28] EU Energy Star: http://eu-energystar.org.

[29] L. Wang, J. Lin, and D. Metzler. Learning to efficiently rank. In *Proc. of SIGIR 2010*, pages 138–145.

[30] 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, 2010.