Efficient Energy Management in Distributed Web Search

Matteo Catena
ISTI-CNR
Pisa, Italy
matteo.catena@isti.cnr.it

Ophir Frieder
Georgetown University
Washington, DC, USA
ophir@ir.cs.georgetown.edu

Nicola Tonellotto
ISTI-CNR
Pisa, Italy
nicola.tonellotto@isti.cnr.it

ABSTRACT

Distributed Web search engines (WSEs) require warehouse-scale computers to deal with the ever-increasing size of the Web and the large amount of user queries they daily receive. The energy consumption of this infrastructure has a major impact on the economic profitability of WSEs. Recently several approaches to reduce the energy consumption of WSEs have been proposed. Such solutions leverage dynamic voltage and frequency scaling techniques in modern CPUs to adapt the WSEs' query processing to the incoming query traffic without negative impacts on latencies.

A state-of-the-art research approach is the PESOS (Predictive Energy Saving Online Scheduling) algorithm, which can reduce the energy consumption of a WSE' single server by up to 50%. We evaluate PESOS on a simulated distributed WSE composed of a thousand of servers, and we compare its performance w.r.t. an industry-level baseline, called PEGASUS. Our results show that PESOS can reduce the CPU energy consumption of a distributed WSE by up to 18% with respect to PEGASUS, while providing query response times which are in line with user expectations.

ACM Reference Format:
Matteo Catena, Ophir Frieder, and Nicola Tonellotto. 2018. Efficient Energy Management in Distributed Web Search. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3269206.3269263

1 INTRODUCTION

High performance query processing is fundamental for the success and the profitability of web search engines (WSEs) [3]. Indeed, WSEs manage an ever growing collection of Web documents and receive billions of queries per day but, at the same time, their users are impatient and expect results for their queries in sub-second times (e.g., 500 ms) [1].

To satisfy such performance requirements, WSEs adopt a distributed architecture, i.e., they are deployed on clusters of thousands of multi-core servers. This architecture ensures that most users will quickly receive their results, keeping them satisfied. However, such many servers consume a significant amount of energy, mostly accountable to the power consumption of their CPUs [2]. The resulting electricity expenditure can hinder the profitability of WSEs, as they can consume tens of megawatts of electric power. Therefore, energy efficiency is an important aspect for the economic successfulness of WSEs. We too focus on energy efficiency and evaluate two energy management approaches using real world data. Our experiments demonstrate considerable gains using a recently proposed research energy management scheduler.

General energy studies are not new, examples including [7, 14] Recent studies show however that users negatively react to large response time, but they can hardly notice response times that are faster than their expectations [1]. Therefore, WSEs can trade-off performance (i.e., longer response times) for lower energy consumptions when this does not affect the user experience. Such trade-offs are feasible by varying the frequency and voltage of CPU cores in WSEs' servers via Dynamic Frequency and Voltage Scaling (DVFS) technologies [13]. Thanks to DVFS, WSEs can save energy by answering queries no faster than necessary. State-of-the-art implementations of this principle are PEGASUS and PESOS.

PEGASUS (Power and Energy Gains Automatically Saved from Underutilized Systems) is a technique that aims at improving the energy efficiency of large scale systems such as WSEs [8]. Experimentally shown, PEGASUS reduces by up to 20% the power consumption of a Google Search production cluster, while keeping its latencies within an acceptable service level objective (SLO). Differently, the PESOS (Predictive Energy Saving Online Scheduling) algorithm is designed to reduce the CPU energy consumption of single servers [4]. Experimentally evaluated, PESOS can reduce the CPU energy consumption of a query processing server by almost 50%, while response times are kept below a desired time threshold.

While PESOS results are significant, their validity is limited, as PESOS has only been tested on a single server configuration. Instead, the de facto standard for WSEs is to rely on a distributed architecture, as PEGASUS correctly assumes. In this work we fill the gap between PESOS and PEGASUS by evaluating the performance of PESOS on a distributed WSE, and comparing PESOS and PEGASUS in terms of energy consumption and their success in meeting response time requirements. To this end, we simulate the behavior of PESOS when deployed on thousands of servers, conducting experiments on the ClueWeb09 (cat. A) corpus and using the MSN2006 query log. Results show that PESOS can reduce the CPU energy consumption of a distributed WSE by up to 18% with respect to PEGASUS, while providing query response times which are in line with user expectations.

2 BACKGROUND

Distributed Search. Given a document collection, a posting list is associated to each term appearing in the collection, containing the list of the documents in which the term occurs. The set of the posting lists for all the terms is called the inverted index. A WSE must manage huge amounts of documents and process billions of

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3269263

queries per day with low latencies. Hence, WSEs partition into smaller *shards* the inverted index used to match user queries. In fact, query processing times depend on the lengths of the posting lists, namely on the number of postings traversed, decompressed, and scored [12]. The posting lists of an index shard are shorter than the corresponding ones in the original inverted index, resulting in reduced processing times.

After partitioning, index shards are assigned to different *shard servers*. When a query is sent to the WSE, it is first received by a *query broker* which dispatches the query to every shard server. Each server computes the query results on its shard independently from the others. These partial results are sent back to the broker, which aggregates them [3, 6]. The aggregated results are the same that would be provided by a single inverted index; since the computation is now distributed across several servers, query processing times are reduced. The query broker collects and aggregates the partial results from the shards, and the final results are sent to the issuing user. The set of shard servers holding all the index shards is an *index replica*.

Since a WSE can receive thousands of queries per second, a single index replica may not be sufficient to deal with such arrival rates. Therefore, WSEs are usually deployed on clusters of servers which host multiple index replicas. By distributing and replicating the inverted index on multiple servers, a WSE can process large volumes of incoming queries with low latencies.

Energy Management. The described distributed and replicated search architectures consume megawatts of electricity, mostly accountable to their CPUs [2]. Yet, relatively few focus on reducing the CPU energy consumption of WSEs without degrading their query latencies. Here we focus on PEGASUS and PESOS as both exploit the CPU energy minimization based on DVFS technology.

PEGASUS [8] is a feedback-based controller designed to manage the power consumption of all the WSE's index replicas. PEGASUS works by dynamically capping the power consumption of the shard servers' CPUs. Since large core frequencies correspond to large power consumption, capping the CPUs' power consumption limits the maximum frequency cores operate. Power capping is performed by constantly monitoring the WSE latencies and by reacting according to a set of rules designed to target a *service level objective* (SLO) latency T. While the actual value of T is not disclosed, it is expressed in terms of the 30-seconds moving average latency. To target T, PEGASUS raises the WSE's power cap when the instantaneous latency Y surpasses T, to avoid SLO violations in the near future. This allows the CPUs' cores to select large frequencies, hence to consume large amount of energy to quickly process queries. Conversely, PEGASUS lowers the power cap when Y is well below T. In that case, SLO violations are unlikely to occur and so CPUs' cores are forced to select small frequencies to save energy. This continuous adjustment of its power cap allows the WSE to meet the SLO while enabling energy savings. All PEGASUS' rules are summarized in Table 1; refer to [8] for further details.

Unlike PEGASUS, PESOS [4] reduces CPU energy consumption of single shard servers, while processing each query within τ ms from its arrival. The algorithm bases its decision on query efficiency predictors (QEPs), which are techniques to estimate the processing time of a query before its processing [10]. PESOS uses QEPs to solve a modified version of the *minimum-energy scheduling problem* (MESP) [16]. In MESP, a set of jobs must be scheduled on a CPU to meet their deadlines and minimize energy consumption of the CPU. Jobs can be preempted and their processing volumes are known, while the CPU speed can vary continuously and is unbounded.

Table 1: PEGASUS' rules as presented in [8]. X is the measured SLO latency (i.e., the 30-seconds moving average latency), Y is the measured instantaneous latency (i.e., the completion time of a query), and T is the target SLO latency.

Input	Action
$X > T$	Set max power, wait 5 minutes
$Y > 1.35T$	Set max power
$Y > T$	Increase power by 7%
$0.85T \leq Y \leq T$	Keep current power
$Y < 0.85T$	Lower power by 1%
$Y < 0.60T$	Lower power by 3%

The MESP always admit a feasible schedule since arbitrary large amounts of work can be performed in infinitesimal time by selecting arbitrary large CPU speeds. Moreover, the MESP can be optimally solved in $O(n^3)$ by using the YDS algorithm proposed by Yao et al. [16]. However, YDS cannot be used directly on search engines since the MESP assumes unconstrained processing speed but the frequencies available on actual CPU cores are discrete and bounded. Moreover, the MESP assumes that job processing volumes are known a priori while query processing volumes may not be [10].

PESOS overcomes these limitations by using the posting as the unit of work associated to a query since the query processing time correlates with the number of postings to evaluate [10]. Consequently, PESOS uses two classes of QEPs: given a query, estimates how many postings are evaluated; and given the number of postings and a core frequency, estimates the query processing time.

PESOS adapts YDS to multi-core shard servers by using such QEPs. It initially replaces query processing volumes with the number of predicted postings to be evaluated, relaying this information to YDS. Then, it translates the cores' speeds returned by YDS into valid core frequencies, by predicting query processing times.

3 DISTRIBUTED WSE SIMULATION

To evaluate the performance of PESOS and PEGASUS at a realistic scale, we simulate a distributed WSE[1]. In doing so, we want to investigate the following research questions (RQs):

(1) Does PESOS help reducing the CPU energy consumption of a distributed WSE?
(2) Does PESOS provide acceptable latencies in a distributed WSE?
(3) How does PESOS compare to PEGASUS in term of both latencies and energy consumption?

To answer RQ1 and RQ3, we simulate the energy consumption (measured in megawatt hours, MWh) of a WSE's CPUs, while we simulate its tail latency (computed 95th percentile of response times distribution) to answer RQ2 and RQ3. To better understand the benefits of PEGASUS and PESOS for WSEs, we will also compare their performance with a WSEs which always operates its CPUs'

[1]The source code is availabe at: https://github.com/catenamatteo/eem-dws-simulator

cores at maximum frequency for the sake of low latencies. We refer to this configuration as PERF.

Simulating the Energy Consumption. A CPU consumes an amount of electric power that depends on the kind of tasks it is performing and on its configuration, i.e., the number of its cores which are actively performing some task and the frequencies at which they operate [2]. For our simulation, we assume that all the WSE's CPUs are Intel i7-4770K, which is the same model used for the experiments in [4]. The i7-4770K has 4 physical cores[2] which expose 15 operational frequencies ranging from 800 MHz to 3.5 GHz. These characteristics result in 3,875 possible CPU configurations in terms of active cores and core frequencies. The simulator maps each of these configurations to their power consumption to accurately estimate the energy consumption of a CPU dedicated to query processing.

We index the ClueWeb09 (cat. B) collection using the Terrier IR platform [9]. We index the collection removing stopwords and applying the Porter stemmer. The resulting inverted index stores document identifiers and term frequencies, compressed with Elias-Fano [15]. The inverted index is kept in main memory by a dedicated server equipped with 32 GB RAM and the i7-4770K processor. This setting is then used to perform the following preliminary experiments. For each possible CPU configuration, we launch a number of instances of the search platform equal to the number of active cores in the configuration. Each search instance is pinned to one of the cores, which operates at the frequency indicated by the CPU configuration. The instances continuously match queries from the first day of the MSN2006 log against the aforementioned inverted index, to retrieve the top 1,000 documents using BM25 and MaxScore. We use Mammut [5] to measure the energy being consumed by the CPU to derive its power consumption.

These preliminary experiments allow us to map each CPU configuration to its power consumption and to use this information in the simulator. Our simulated CPU consumes 0.8 Watts when idle, and up to 34.2 Watts when all its cores are busy processing queries at 3.5 Ghz.

To validate our approach, we simulated the processing of the whole second day of the MSN2006 query log against the ClueWeb09 (cat. B) inverted index, on a single server with a i7-4770K CPU and 32 GB RAM. The simulated server adopts PESOS to reduce its CPU energy consumption. The same experiment was conducted in the real world, and we find that our simulation underestimates by just ~1% the actual energy consumption of the server's CPU.

Simulating the Latency. We simulate the WSE tail rather than mean latency as tail latency is considered a better performance indicator [6]. In particular, for comparison purposes we chose the 95th percentile latency to mirror prior efforts [4, 8, 11].

The data used in our simulation are collected from real-world experiments carried out using Terrier on the same server described earlier. In this case, we use the ClueWeb09 (cat. A) collection, whose Web pages are organized into five different folders (B, A2, A3, A4, and A5) that we use as a form of document partitioning to perform distributed search. In practice, we consider ClueWeb09 (cat. A) to be composed by five partitions (the folders), each containing ~50 millions Web pages. We use Terrier to index each of these partitions independently, in the same way described earlier.

Since the resulting inverted index is composed by five shards, each index replica of our simulated WSE manages five shard servers. Lo et al. do not report the exact number of index replicas in the Google Search production cluster used for their experiments [8]. They rather affirm that the cluster contains "thousands of servers", and we speculate it has at least 1,000 servers. Therefore, we adopt the same number of shard servers in our simulator, resulting in the simulation of a distributed WSE with 200 index replicas.

To exercise our simulated WSE, we use the second day of query arrivals from the MSN2006 log. This reports only a small fraction of the actual arrivals, which can be efficiently served with just one index replica. Therefore, we decided to multiply the number of queries received every second by a factor 200, i.e., for every query arrival in the original query log we simulate 200 concurrent query arrivals. For each of these arrivals, we generate a simulated query according to a *query template*. A query template is a representation of an actual query, which reports its number of terms and its processing time on every index shard and at every frequency. We build our query templates by randomly sampling 10,000 unique queries from the second day of the MSN2006 log. We process these queries using Terrier as described earlier. To build its query template, every query is processed on each of the five index shards and at each of the 15 core frequencies (i.e., 75 times).

Configuring PEGASUS and PESOS. Originally, PEGASUS expresses the target SLO latency T in terms of the 30-seconds moving average latency metric. Since we focus on the 95th-tile percentile latency, in our simulator, we modified PEGASUS' rules in Table 1 to express the SLO as the 30-seconds moving 95th-percentile. The target SLO is set to 500 ms, according to [1]. Similarly, we also impose the time budget τ of PESOS to 500 ms. Additionally, PESOS requires a training phase to generate its QEPs, as in [10]. We train the QEPs as described in [4], i.e., we train different QEPs for queries of different lengths. For each length, we used 10,000 unique queries, randomly sampled from the first day of the MSN2006 log. We train QEPs for queries with 1 up to 5 query terms, while longer queries (6+ terms) are managed by a common QEPs. In total, we use 60,000 queries for the training phase of PESOS. Once QEPs are trained, we estimate the number of postings to score and the processing times at each core frequency of the 10,000 query templates used in the simulation. Predictions are performed offline since they take less than 0.2 ms on average, i.e., they are unlikely to affect the simulation times.

4 EXPERIMENTAL RESULTS

To answer RQ2 and RQ3, we initially discuss the latencies of our simulated WSE in its various configurations. Then, we analyze the CPU energy consumption of PEGASUS and PESOS w.r.t. PERF to answer RQ1 and RQ3.

In Figure 1 we observe that both PEGASUS and PESOS are closer than PERF to the 500 ms SLO. PERF exhibits a tail latency of ~350 ms at the cost of a large energy consumption. While PESOS tail latencies are closer than PEGASUS' ones to the target SLO, we also observe that PESOS leads to small latency violations in the early and late hours. This happens because the WSE receives fewer queries during nighttime than during daytime; therefore PESOS

[2]We do not exploit here hyperthreading, reflecting the experimental setup in [4].

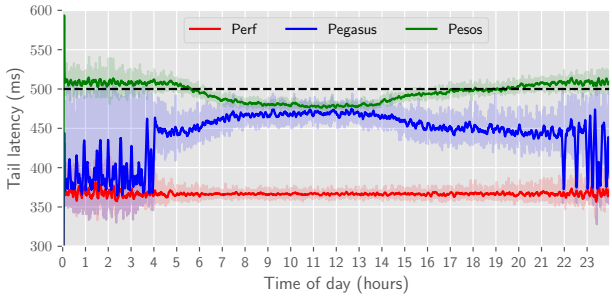

Figure 1: The 30-seconds moving 95th-%tile latency of PERF, PEGASUS, and PESOS, both per second and with a order 3 Savitzky-Golay smoothig over a ten minutes time window.

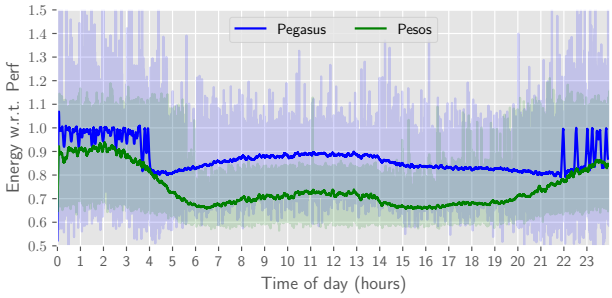

Figure 2: The CPU energy consumption of PEGASUS and PESOS w.r.t. PERF, both per second and with a order 3 Savitzky-Golay smoothing over a ten minutes time window.

tends to select small core frequencies. This results in previously unobserved latency violations [4], but which emerge at scale due to the variability of response times across different shard servers [6]. Nevertheless, such limited violations (on average 509 ms) do not negatively affect the user experience [1].

In the same periods of the day, PEGASUS processes queries much faster than necessary, likely because PEGASUS *"is a conservative policy that aims to slowly reduce the power limit without causing any SLO violations along the way"* [8]. Therefore, PEGASUS fails to capture full advantage of small workloads to save energy, to not hinder the system responsiveness. This is why the CPU energy consumption of PEGASUS is similar to PERF in early and late hours, as shown in Figure 2. Energy savings up to ~20% w.r.t. PERF are observable during the rest of the day, confirming the results in [8].

Conversely, PESOS can save ~10% of CPU energy consumption w.r.t. PERF early and late in the day. However, this comes at the cost of small latency violations as shown in Figure 1. During the rest of the day, PESOS remarkably reduces the CPU energy consumption by ~30% w.r.t. PERF, while keeping the WSE's tail latency just below 500 ms. In fact, query workload is intense during midday, and PESOS correctly selects large core frequencies in such situation [4].

Given these results, we can conclude that PESOS helps reducing the CPU energy consumption of a distributed WSE (RQ1). In our simulations, the CPUs in a day consume 254.02 MWh using PERF, while with PESOS the consumption reduces to 179.26 MWh (-29%). The same CPUs consume 218.40 MWh with PEGASUS, meaning that PESOS consumes 18% less energy than PEGASUS (RQ3). However, such energy savings comes at the cost of negligible latency violations when workload is scarce (RQ2), while PEGASUS never violates the target SLO (RQ3).

5 CONCLUSIONS

We evaluated the performance of PESOS, an energy-saving strategy designed for WSEs' single servers, when deployed on a distributed infrastructure. We compared PESOS to PEGASUS, an industry-level baseline. Both strategies adapt the CPUs core frequencies to the incoming query traffic exploiting DVFS technologies. We simulated a distributed WSE deployed on a thousand servers, using the ClueWeb09 document collection and the MSN query log. The open-sourced simulation has been finely tuned to reproduce real-world measurements both in terms of tail latencies and energy consumption (~1% deviations). Our results showed that PESOS reduces the CPU energy consumption of a distributed WSE by up to 18% w.r.t. PEGASUS, at the cost of negligible latency violations (less than 2% on average) during low workloads periods.

ACKNOWLEDGMENTS

This paper is partially supported by the BIGDATAGRAPES (grant agreement N°780751) project that received funding from the European Union's Horizon 2020 research and innovation programme under the Information and Communication Technologies work programme. We thank Stefano Ceccotti for his help in the early stages of this work.

REFERENCES

[1] Ioannis Arapakis, Xiao Bai, and B. Barla Cambazoglu. 2014. Impact of Response Latency on User Behavior in Web Search. In *Proc. SIGIR*. 103–112.
[2] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. 2013. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines* (2nd ed.). Morgan & Claypool Publishers.
[3] B. Barla Cambazoglu and Ricardo A. Baeza-Yates. 2015. *Scalability Challenges in Web Search Engines*. Morgan & Claypool Publishers.
[4] Matteo Catena and Nicola Tonellotto. 2017. Energy-Efficient Query Processing in Web Search Engines. *IEEE TKDE* 29, 7 (2017), 1412–1425.
[5] Daniele De Sensi, Massimo Torquati, and Marco Danelutto. 2017. Mammut: High-level management of system knobs and sensors. *SoftwareX* 6 (2017), 150 – 154.
[6] Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. *Commun. ACM* 56, 2 (2013), 74–80.
[7] Enver Kayaaslan, B. Barla Cambazoglu, Roi Blanco, Flavio P. Junqueira, and Cevdet Aykanat. 2011. Energy-price-driven Query Processing in Multi-center Web Search Engines. In *Proc. SIGIR*. 983–992.
[8] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. 2014. Towards Energy Proportionality for Large-scale Latency-critical Workloads. In *Proc. ISCA*. 301–312.
[9] Craig Macdonald, Richard McCreadie, Rodrygo LT Santos, and Iadh Ounis. 2012. From puppy to maturity: Experiences in developing Terrier. *Proc. OSIR at SIGIR* (2012), 60–63.
[10] Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2012. Learning to Predict Response Times for Online Query Scheduling. In *Proc. SIGIR*. 621–630.
[11] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. 2011. Power Management of Online Data-intensive Services. In *Proc. ISCA*. 319–330.
[12] Alistair Moffat, William Webber, Justin Zobel, and Ricardo Baeza-Yates. 2007. A Pipelined Architecture for Distributed Text Query Evaluation. *Information Retrieval* 10, 3 (2007), 205–231. Kluwer Academic Publishers.
[13] David C. Snowdon, Sergio Ruocco, and Gernot Heiser. 2005. Power Management and Dynamic Voltage Scaling: Myths and Facts. In *Proc. Workshop on Power Aware Real-time Computing*.
[14] Amin Teymorian, Ophir Frieder, and Marcus A. Maloof. 2013. Rank-energy Selective Query Forwarding for Distributed Search Systems. In *Proc. CIKM*. 389–398.
[15] Sebastiano Vigna. 2013. Quasi-succinct indices. In *Proc. WSDM*. 83–92.
[16] F. Yao, A. Demers, and S. Shenker. 1995. A scheduling model for reduced CPU energy. In *Proc. FOCS*. 374–382.