Dogo Rangsang Research Journal
ISSN : 2347-7180
UGC Care Group I Journal
Vol-9 Issue-3 Sept - Dec 2019

An Efficacious Query Processing Approach with Predictive Energy Saving using Online Scheduling

Maragani Harinadha[1], Chinthala Shekhar[2]

[1,2]*Assistant Professor, Department of Computer Science and Engineering*
[1,2]*Malla Reddy Engineering College (A), Hyderabad, Telangana, India.*

Abstract

Web search engines are composed by thousands of query processing nodes, i.e., servers dedicated to process user queries. Such many servers consume a significant amount of energy, mostly accountable to their CPUs, but they are necessary to ensure low latencies, since users expect sub-second response times (e.g., 500 ms). However, users can hardly notice response times that are faster than their expectations. Hence, we propose the Predictive Energy Saving Online Scheduling Algorithm (PESOS) to select the most appropriate CPU frequency to process a query on a per-core basis. PESOS aim at process queries by their deadlines and leverage high-level scheduling information to reduce the CPU energy consumption of a query processing node. PESOS base its decision on query efficiency predictors, estimating the processing volume and processing time of a query. We experimentally evaluate PESOS upon the TREC ClueWeb09B collection and the MSN2006 query log. Results show that PESOS can reduce the CPU energy consumption of a query processing node up to ∼48% compared to a system running at maximum CPU core frequency. PESOS outperform also the best state of the-art competitor with a ∼20% energy saving, while the competitor requires a fine parameter tuning and it may incur in uncontrollable latency violations.

Keywords: *Energy consumption, CPU Dynamic Voltage and Frequency Scaling, Web search engines.*

1. Introduction

Web search engines continuously crawl and index an immense number of Web pages to return fresh and relevant results to the users‟ queries. Users‟ queries are processed by query processing nodes, i.e., physical servers dedicated to this task. Web search engines are typically composed by thousands of these nodes, hosted in large data centers which also include infrastructures for telecommunication, thermal cooling, fire suppression, power supply, etc [1]. This complex infrastructure is necessary to have low tail latencies (e.g., 95-th percentile) to guarantee that most users will receive results in sub-second times (e.g., 500 ms), in line with their expectations [2]. At the same time, such many servers consume a significant amount of energy, hindering the profitability of the search engines and raising environmental concerns. In fact, data centers can consume tens of megawatts of electric power [1] and the related expenditure can exceed the original investment cost for a data center [3]. Because of their energy consumption, datacentres are responsible for the 14% of the ICT sector carbon dioxide emissions [4], which are the main cause of global warming. For this reason, governments are promoting codes of conduct and best practices [5], [6] to reduce the environmental impact of data centers.

Since energy consumption has an important role on the profitability and environmental impact of Web search engines, improving their energy efficiency is an important aspect. Noticeably, users can hardly notice response times that are faster than their expectations [2]. Therefore, to reduce energy consumption, Web search engines should answer queries no faster than user expectations. In this work, we focus on reducing the energy

consumption of servers‟ CPUs, which are the most energy consuming components in search systems [1]. To this end, Dynamic Frequency and Voltage Scaling (DVFS) technologies [7] can be exploited. DVFS technologies allow to vary the frequency and voltage of the CPU cores of a server, trading off performance (i.e., longer response times) for lower energy consumptions. Several power management policies leverage DVFS technologies to scale the frequency of CPU cores accordingly to their utilization [8], [9]. However, core utilization-based policies have no mean to impose a required tail latency on a query processing node. As a result, the query processing node can consume more energy than necessary in providing query results faster than required, with no benefit for the users.

In this work we propose the Predictive Energy Saving On-line Scheduling algorithm (PESOS), which considers the tail latency requirement of queries as an explicit parameter. Via the DVFS technology, PESOS select the most appropriate CPU frequency to process a query on a per-core basis, so that the CPU energy consumption is reduced while respecting a required tail latency. The algorithm bases its decision on query efficiency predictors rather than core utilization. Query efficiency predictors are techniques to estimate the processing time of a query before its processing. They have been proposed to improve the performance of a search engine, for instance to take decision about query scheduling [10] or query processing parallelization [11], [12]. However, to the best of our knowledge, query efficiency predictor has not been considered for reducing the energy consumption of query processing nodes

We build upon the approach described in [10] and propose two novel query efficiency predictor techniques: one to estimate the number of postings that must be scored to process a query, and one to estimate the response time of a query under a particular core frequency given the number of postings to score. PESOS exploit these two predictors to determine which is the lowest possible core frequency that can be used to process a query, so that the CPU energy consumption is reduced while satisfying the required tail latency. As predictors can be inaccurate, in this work we also propose and investigate a way to compensate prediction errors using the root mean square error of the predictors.

We experimentally evaluate PESOS upon the TREC ClueWeb09 corpus and the query stream from the MSN2006 query log. We compare the performance of our approach with those of three baselines: perf [8], which always uses the maximum CPU core frequency, power [8], which throttles CPU core frequencies according to the core utilizations, and cons [13], which performs frequency throttling according to the query server utilization. PESOS, with predictors correction, is able to meet the tail latency requirements while reducing the CPU energy consumption from ∼24% up to ∼44% with respect to perf and up to ∼20% with respect to cons, which however incurs in uncontrollable latency violations. Moreover, the experiments show that energy consumption can be further reduced by PESOS when prediction correction is not used, but with higher tail latencies.

2. Related Work

In the past, a large part of a data center energy consumption was accounted to inefficiencies in its cooling and power supply systems. However, Barroso et al. [1] report that modern data centers have largely reduced the energy wastage of those infrastructures, leaving little room for further improvement. On the contrary, opportunities exist to reduce the energy consumption of the servers hosted in a data center. In particular, our work focuses on the CPU power management of query processing nodes, since the CPUs dominate the energy consumption of physical servers dedicated to search tasks. In fact, CPUs can use up to 66% of the whole energy consumed by a query processing node at peak utilization [1].

Modern CPUs usually expose two energy saving mechanism, namely C-states and P-states. C-states represent CPU cores idle states and they are typically managed by the operating system [14]. C0 is the operative state in which a CPU core can perform computing tasks. When idle periods occur, i.e., when there are no computing tasks to perform, the core can enter one of the other deeper C-states and become inoperative. However, Web search engines process a large and continuous stream of queries. As a result, query processing nodes are rarely inactive and experience particularly short idle times. Consequently, there are little opportunities to exploit deep C-states, reducing the energy savings provided by the C-states in a Web search engine system [15], [16].

When a CPU core is in the active C0 state, it can operate at different frequencies (e.g., 800 MHz, 1.6 GHz, 2.1 GHz, . . .). This is possible thanks to the Dynamic Frequency and Voltage Scaling (DVFS) technology [7] which permits to adjust the frequency and voltage of a core to vary its performance and power consumption. In fact, higher core frequencies mean faster computations but higher power consumption. Vice versa, lower frequencies lead to slower computations and reduced power consumption. The various configurations of voltage and frequency available to the CPU cores are mapped to different P-states and are managed by the operating system. For instance, the intel_pstate driver [8] controls the P-states on Linux systems and can operate accordingly to two different policies, namely perf and power. The perf policy simply uses the highest frequency to process computing tasks. Instead, power selects the frequency for a core according to its utilization. When a core is highly utilized, power selects a high frequency. Conversely, it will select a lower frequency when the core is lowly utilized.

However, Lo et. al [15] argue that core utilization is a poor choice for managing the cores frequencies of query processing nodes. In fact, the authors report an increase of query response times when core utilization-based policies are used in a Web search engine. For such reason, Catena et al. [13] propose to control the frequency of CPU cores based on the utilization of the query processing node rather than on the utilization of the cores. The utilization of a node is computed as the ratio between the query arrival rate and service rate. Then, they propose the cons policy which throttles the frequency of the CPU cores when the utilization of the node is above or below certain thresholds (e.g., 80% and 20%, respectively). The frequency is selected so to produce a desirable utilization level (e.g., 70%). Similarly, in our work we control the CPU cores frequencies of a query processing node using information related to the query processing activity rather than to the CPU cores utilization. To this end, we build our approach on top of the acpi_cpufreq driver [9]. This driver allows applications to directly manage the CPU cores frequency, instead of relying on the operative systems.

Query efficiency predictors (QEPs) are techniques that estimate the execution time of a query before it is actually processed. Knowing in advance the execution time of queries permits to improve the performance of a search engine. Most QEPs exploit the characteristics of the query and the inverted index to pre-compute features to be exploited to estimate the query processing times. For instance, Macdonald et al. [10] propose to use term-based features (e.g., the inverse document frequency of the term, its maximum relevance score among others) to predict the execution time of a query. They exploit their QEPs to implement on-line algorithms to schedule queries across processing node, in order to reduce the average query waiting and completion times. The works [11], [12], instead, address the problem to whether parallelize or not the processing of a query. In fact, parallel processing can reduce the execution time of long-running queries but provides limited benefits when dealing with short-running ones. Both the works propose QEPs to detect long-running queries. The processing of the query is parallelized only if their QEPs detect the query as a long-running one. Rather than combining term-based features, Wu et al. [17] propose to analytically model the query processing stages and to use such model to predict the execution time of queries.

In our work, we modify the QEPs described in [10] to develop our algorithm for reducing the energy consumption of a processing node while maintaining low tail latencies.

3. Proposed Methodology

In the following, we introduce the operative scenario of a query processing node (Sec. 3.1), we formalize the general minimum-energy scheduling problem and we shortly present the state-of-the-art algorithm to solve it offline (Sec. 3.2), and we discuss the issues of this offline algorithm in our scenario (Sec. 3.3).

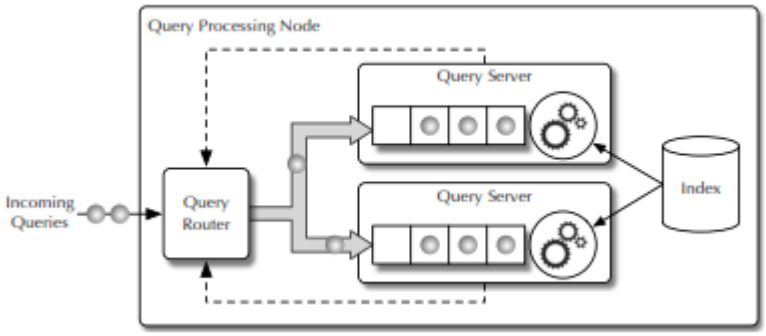

Figure 1. The architecture of query processing node.

3.1 Operative scenario

A query processing node is a physical server composed by several multi-core processors/CPUs with a shared memory which holds the inverted index. The inverted index can be partitioned into shards and distributed across multiple query processing nodes. In this work, we focus on reducing the CPU energy consumption of single query processing nodes, independently of the adopted partition strategy. In the following, we assume that each query processing node holds an identical replica of the inverted index [18]. A query server process is executed on top of each of the CPU core of the processing node (see Figure 1). All query servers access a shared inverted index held in main memory to process queries. Each query server manages a queue, where the incoming queries are stored. The first query in the queue is processed as soon as the corresponding CPU core is idle. The queued queries are processed following the first come first served policy. The number of queries in a query server's queue represents the server load. Queries arrive to the processing node as a stream S = {q1, . . . , qn}. When a query reaches the processing node it is dispatched to a query server by a query router. The query router dispatches an incoming query to the least loaded query server, i.e., to the server with the smallest number of enqueued queries. Alternatively, the query processing node could have a single query queue and dispatch queries from the queue to idle query servers. In this work, we use a queue for each query servers since a single queue will not permit to take local decisions about the CPU core frequency to use for the relative query server. A similar queue-per core architecture is assumed in [19], to schedule jobs across CPU cores to minimize the CPU energy consumption, and in [10].

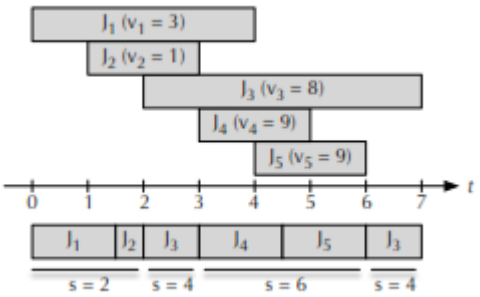

Figure 2. The example of YDS scheduling, (top) input jobs, (bottom) resulting optimal schedule with CPU speeds *s*.

3.2 The minimum-energy scheduling problem

Consider the following scenario, where a single core CPU must execute a set $J = \{J_1, \ldots, J_n\}$ of generic computing jobs rather than queries. Jobs must be executed over a time interval $[t_1, t_0]$. Each job J_i has an arrival time a_i and an arbitrary deadline d_i which are known a priori. Moreover, each job J_i has a processing volume v_i, i.e., how much work it requires from the CPU, and jobs can be pre-empted. The CPU can operate at any processing speed $s \in \mathbb{R}^+$ (in time units per unit of work) and its power consumption is a convex function of the processing speed, e.g., $P(s) = s^\gamma$ with $\gamma > 1$ [7]. Jobs in J must be scheduled on the CPU. A schedule is a pair of functions $S = (\psi, \phi)$ denoting, respectively, the processing speed and the job in execution, both at time t

A schedule is feasible if each job in J is completed within its deadline. The minimum-energy scheduling problem (MESP) aims at finding a feasible schedule such that the total energy consumption is minimized, i.e.,

$$\arg\min_{S=(\psi,\phi)} E(S) = \int_{t_0}^{t_1} P\big(\psi(t)\big)dt$$

Figure 2 shows an example for YDS. Input jobs are illustrated in the upper part of the picture. The left end of a box indicates the arrival time of the job, while the right end indicates its deadline. Processing volumes for the jobs are reported inside the relative boxes. The bottom part of the picture illustrates the optimal solution provided by YDS. The picture shows the order in which the jobs are scheduled, their start and end time, and the processing speeds s used for each job. Note that J_3 is executed over two different time intervals, as it is pre-empted to schedule J_4 and J_5, which have a higher joint intensity.

3.3 Issues with YDS

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

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

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

3) YDS schedules job using processing speeds (defined as units of work per time unit). The speed value is continuous and unbounded (i.e., the speed can be indefinitely large). However, the frequencies available to CPU cores are generally discrete and bounded. For such reasons, in the following Section we modify YDS in order to exploit it in a search engine.

4. Conclusions

In this paper we proposed the Predictive Energy Saving Online Scheduling (PESOS) algorithm. In the context of Web search engines, PESOS aim to reduce the CPU energy consumption of a query processing node while imposing a required tail latency on the query response times. For each query, PESOS select the lowest possible CPU core frequency such that the energy consumption is reduced, and the deadlines are respected. PESOS select the right CPU core frequency exploiting two different kinds of query efficiency predictors (QEPs). The first QEP estimates the processing volume of queries. The second QEP estimates the query processing times under different core frequencies, given the number of postings to score. Since QEPs can be inaccurate, during their training we recorded the root mean square error (RMSE) of the predictions. In this work, we proposed to sum the RMSE to the actual predictions to compensate prediction errors. We then defined two possible configurations for PESOS: time conservative, where prediction correction is enforced, and energy conservative, where QEPs are left unmodified.

References

[1] L. A. Barroso, J. Clidaras, and U. Ho¨lzle, The Data center as a Computer: An Introduction to the Design of Warehouse-Scale Machines,2nded. Morgan & Claypool Publishers, 2013.

[2] I. Arapakis, X. Bai, and B. B. Cambazoglu, “Impact of response latency on user behavior in web search,” in Proc. SIGIR, 2014, pp.103–112.

[3] U.S. Department of Energy, “Quick start guide to increase data center energy efficiency,” 2009. [Online]. Available: http://goo.gl/ovDP26

[4] The Climate Group for the Global e-Sustainability Initiative, “Smart 2020: Enabling the low carbon economy in the information age,” 2008. [Online]. Available: http://goo.gl/w5gMXa

[5] European Commission - Joint Research Centre, “The European Code of Conduct for Energy Efficiency in Data Centre.”[Online]. Available: http://goo.gl/wmqYLQ

[6] U.S. Department of Energy, “Best Practices Guide for Energy-Efficient Data Center Design.” [Online]. Available: http://goo.gl/pikFFv

[7] D. C. Snowdon, S. Ruocco, and G. Heiser, “Power Management and Dynamic Voltage Scaling: Myths and Facts,” in Proc. of Workshop on Power Aware Real-time Computing, 2005.

[8] The Linux Kernel Archives, “Intel P-State driver.” [Online]. Available: https://goo.gl/w9JyBa

[9] D. Brodowski, “CPU frequency and voltage scaling code in the Linuxkernel.” [Online]. Available: https://goo.gl/QSkft2

[10] C. Macdonald, N. Tonellotto, and I.Ounis, “Learning to predict response times for online query scheduling,” in Proc. SIGIR, 2012, pp.621–630.

[11] M. Jeon, S. Kim, S.-w. Hwang, Y. He, S. Elnikety, A. L. Cox, and S. Rixner, “Predictive parallelization: Taming tail latencies in web search,” in Proc. SIGIR, 2014, pp.253–262.

[12] S. Kim,Y. He, S.-w. Hwang, S. Elnikety, and S. Choi, “Delayed dynamic-selective (dds) prediction for reducing extreme tail latency in web search,” in Proc.WSDM,2015,pp.7–16.

[13] M. Catena, C. Macdonald, and N. Tonellotto, “Load-sensitive cpu power management for web search engines,” in Proc. SIGIR, 2015, pp.751–754.

[14] V. Pallipadi, S. Li, and A. Belay, “cpuidle: Do nothing, efficiently,” in Proc. Linux Symposium, vol.2, 2007, pp.119–125.

[15] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in Proc. ISCA, 2014, pp. 301–312.

[16] D. Meisner, C. M. Sadler, L. A. Barroso, W. -D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in Proc. ISCA, 2011, pp.319–330.

[17] H. Wu and H. Fang, "Analytical performance modelling for top-k query processing," in Proc. CIKM, 2014, pp.1619–1628.

[18] A. Freire, C. Macdonald, N. Tonellotto, I. Ounis, and F. Cacheda, "Hybrid query scheduling for a replicated search engine,"inProc.ECIR,2013,pp.435–446.

[19] S. Albers, F. Mu¨ller, and S. Schmelzer, "Speed scaling on parallel processors," in Proc. SPAA, 2007, pp.289–298.

