Information Processing and Management 000 (2016) 1–22

Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/ipm

Reducing hardware hit by queries in web search engines

Marcelo Mendoza[a,*], Mauricio Marín[b], Verónica Gil-Costa[c], Flavio Ferrarotti[d]

[a] Universidad Técnica Federico Santa María, Santiago, Chile
[b] Universidad de Santiago de Chile, Santiago, Chile
[c] CONICET, Universidad Nacional de San Luis, Argentina
[d] Software Competence Center Hagenberg, Austria

ARTICLE INFO

Article history:
Received 5 December 2015
Revised 19 April 2016
Accepted 23 April 2016
Available online xxx

Keywords:
Query routing
Distributed information retrieval
Incremental learning

ABSTRACT

In this paper, we introduce a new collection selection strategy to be operated in search engines with document partitioned indexes. Our method involves the selection of those document partitions that are most likely to deliver the best results to the formulated queries, reducing the number of queries that are submitted to each partition. This method employs learning algorithms that are capable of ranking the partitions, maximizing the probability of recovering documents with high gain. The method operates by building vector representations of each partition on the term space that is spanned by the queries. The proposed method is able to generalize to new queries and elaborate document lists with high precision for queries not considered during the training phase. To update the representations of each partition, our method employs incremental learning strategies. Beginning with an inversion test of the partition lists, we identify queries that contribute with new information and add them to the training phase. The experimental results show that our collection selection method favorably compares with state-of-the-art methods. In addition our method achieves a suitable performance with low parameter sensitivity making it applicable to search engines with hundreds of partitions.

1. Introduction

The Web has significantly expanded forcing search engines to handle document collections with millions of websites and web pages. Search engines use collection partition strategies enabling operations on small subcollections and render their processing feasible for both indexing and retrieving operations. A prevalent trend is to consolidate thematic collections consisting of documents that address specific topics. This trend is known as vertical search and requires adequate collection partitioning strategies.

A search engine can receive an enormous amount of queries on a daily basis. For this reason, one or more machines are exclusively devoted to the task of receiving and processing these queries. These machines are referred to as brokers. Brokers route queries to machines that have documents and local inverted indexes. Two routing methods are employed: the first routing method, known as a broadcast, sends a query to all partitions; the second method, known as a multicast, only sends a query to some partitions. In the latter case, a selective partition routing method is required (query routing). These methods generally employ descriptions of the contents of each partition in order to determine their similarity to the

* Corresponding author. Tel.: +56223037213.
E-mail addresses: marcelo.mendoza@usm.cl, mmendoza@inf.utfsm.cl (M. Mendoza), mauricio.marin@usach.cl (M. Marín), gvcosta@unsl.edu.ar (V. Gil-Costa), flavio.ferraroti@scch.at (F. Ferrarotti).

http://dx.doi.org/10.1016/j.ipm.2016.04.008
0306-4573/

query. Depending on this information, they select the most promising partitions for processing. This query routing method is referred to as collection selection.

Two methods for collection partitioning are available. The first method, known as term or horizontal partitioning, consists of spliting the vocabulary. This strategy enables routing the inverted index of an entire collection and indexing each partition of the vocabulary in different machines. The second strategy is known as document or vertical partitioning and consists of dividing a document collection in disjoint subsets. This strategy by placing each document subset in a machine along a local inverted index. These strategies enable to process queries in multicast mode, which reduces the number of partitions that are required to process each query. This approach results in lower computational costs at the partition level and improvements in a complete system, in terms of query throughput. In this paper, we employ this scenario, assuming a vertical partition of the collection, and applying a collection selection method that is capable of identifying a subset of promising partitions in the broker to process a given query.

We propose a new collection selection method. Our method builds vector representations of each document partition over the query term space. The construction of these representations is performed using a learning algorithm based on logistic regression, which is capable of finding regions in the query space where an information recovery reward function is maximized. As a reward function, we employ the recall of each partition over the exact top-k document list and the normalized discount cumulative gain (NDCG), which considers not only the fraction of relevant and recovered documents but also their ranking. The learning algorithm operates over all collection responses, which are calculated using a broadcast and correspond to the gold standard of the learning process. Beginning with this information, we can generalize to new queries that share terms with the set of queries initially employed to build each partition representation. When detecting new queries, our method is capable of determining whether they should be included in the learning phase. The decision is simply attained by comparing the ranking inversions in the processing list of broadcast versus multicast. Using a statistical criterion, we can determine if both lists were generated using the same conditions or were randomly jointly generated.

Our proposal's main strength is the possibility of providing control to precision versus workload tradeoff. By implementing our method in search engines with vertical partitioning, the method can be employed in high-query traffic conditions without losing response quality.

The main contributions of this study are as follows:

- The construction of representations of each partition's content, without document content: We employ the query terms, which enable us to model the aspect of each partition from the user's point of view. This approach enables us to conduct a learning process over a term space with lower dimensionality and higher representation, reducing the computational costs associated with the training phase of the collection selection method;
- The modeling of the processor lists in the learning process using recall or the NDCG as a reward function for the process: This step enables the incorporation of a reward function, whose optimization is required within the collection selection method; and
- The incorporation of new queries into the learning process using information novelty favoring the elimination of redundancy avoiding overfitting.

This paper is organized as follows. In Section 2, we perform a review of the literature, identify a previous study whose proposal is the closest to ours, and compare our own results with results of the selected study. In Section 3, we describe the necessary background for constructing a vertically partitioned search engine, on which our proposal is constructed. In Section 4, we introduce our learning-based collection selection method. In Section 5, we present our incremental learning strategy. In Section 6, we present experiments using real data. We conclude in Section 7, where we highlight this study's main achievements.

2. Related work

2.1. Data distribution strategies

When document's distribution is done following an architecture partitioned by terms, each processor stores a portion of the global index representing a fraction of the vocabulary. The problem of identifying an appropriate partition for the global index allowing a balanced workload is complex. Moffat, Webber, and Zobel (2006) modelled the term partitioning as a *bin packing* problem. There, each bin represents a partition where each term is an object to be placed into the bin. By using the frequency of each query term registered in a log file, they associate each term with a weight proportional to its frequency and with the length of its respective posting list. Through the use of these weights, a function of the query processing costs is constructed. Then, the bin packing problem is solved by minimizing the cost function. Using a cluster of eight processors, the authors show that it is possible to improve query throughput by up to 30% when compared to a centralized system.

Lucchese, Orlando, Perego, and Silvestri (2007) also modelled the term partitioning as a bin packing problem. Unlike the work of Moffat et al., they seek to model queries instead of isolated terms in each of the bins. To do this, they analyzed a query log determining the term co-occurrence in queries adding this variable to the cost function. After minimizing the cost function, they can determine good solutions for partitioning the global index, improving results based on isolated terms.

Cambazoglu, Kayaaslan, Jonassen, and Aykanat (2013) proposed a term-based index partitioned scheme by using hypergraph partitioning. The goal is to minimize the communication overhead while achieving good computational load balance

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

among servers. The proposed scheme distributes inverted lists such that the lists that are likely to be accessed together are assigned to the same index servers. Accesses to inverted lists are represented by a hypergraph. Results showed a reduction on query response time regarding bin packing-based strategies.

Jonassen and Bratsberg (2014) proposed to use the idea of Max-Score pruning within pipelined query processing to improve query throughput for a term partition-based scheme. Their proposal achieve good results however the authors recognize that document-wise partitioning still presents better performance.

One of the main disadvantages of term partitioning strategies is that they require a global index, which limits scalability. This limitation is dealt with success by document partitioning strategies. In this type of strategies, documents are first distributed and then a local index is constructed in each processor.

A simple approximation for document partitioning consists of distributing them randomly into processors. However, Badue, Baeza-Yates, Ribeiro-Neto, Ziviani, and Ziviani (2007) show that a homogeneous documents distribution into processors does not guarantee load balance. Then, document partitioning strategies address this problem by identifying groups of similar documents, placing them into the same processors. In this way, it is expected that given a query only a few processors achieve relevant results. Then, different queries tend to be processed by different processors favoring load balancing.

The state of the art shows that several strategies for grouping similar documents have been tried. Broder, Glassman, Manasse, and Zweig (1997) propose doing a syntactic grouping of documents on the Web. For each document, they determine fixed length term sequences that they call *shingles*. Using a compact shingle representation, they calculate a similarity measure among document pairs based on the intersection between the collections of shingles that represent each document. Then, using standard clustering strategies they are able to determine groups of similar documents. Puppin (2007) showed that by grouping similar documents extracted from the Irudiko library, a document partitioning strategy based on shingles allowed them to obtain improvements over a random document distribution.

Another way of grouping documents consists of ordering their URLs lexicographically, partitioning document lists into fixed-length blocks. Then, each block can be considered as a group of documents. URL sorting was used by Silvestri (2007) to probe that it is possible to obtain a better compression ratio in posting lists. This strategy involves low computational costs, making it an appropriate method for large-scale systems.

Puppin, Silvestri, and Laforenza (2006) represent documents using a *query-vector model*. After processing query logs, each document is represented using the list of queries from which they were selected. In this way, a matrix is constructed in such a way where each row represents a query and each column represents a document. Each entry in the matrix is calculated by using the score that the document achieved in the list of query results. The authors propose grouping documents using the co-clustering algorithm introduced by Dhillon, Mallela, and Modha (2003) determining document groups relevant to the same queries. The authors provide an implementation of the co-clustering algorithm (Puppin & Silvestri, 2007) that allows large quantities of documents and queries to be processed, favoring their use in large-scale systems.

Ma, Chung, and Chen (2011) proposed a clustered search engine based on statistics where the average query processing time is estimated taking into account the popularities of keyword terms. It is an approach that tries to eliminate the communication overhead in document-wise distributed inverted indexes. The partitioning algorithm is static and works off-line. The evaluation is performed in terms of both performance and storage cost.

An hybrid distribution approach is proposed by Kucukyilmaz, Turk, and Aykanat (2012), who presented a term-based partitioned inverted index construction algorithm which starts from a document-based partitioned collection. The proposal is based on a bucketing scheme which avoids the need of creating a global vocabulary. Buckets are used to randomly group inverted lists (a hash function is used to assign the terms into buckets). A host processor generates a mapping function using the sizes of their term buckets allocated in each processor to achieve load balance and reduce communication.

2.2. Collection selection

Collection selection techniques represent groups of similar documents using the text of the documents that they contain. Along this research line, Gravano, Garcia-Molina, and Tomasic (1994) introduced GLOSS (Glossary of Servers Server), an architecture for distributed documentary databases which includes a Boolean information retrieval model in the broker to route queries to a subset of document clusters. A similar approach known as CORI (Collection Retrieval Inference) was introduced by Callan, Lu, and Croft (1995). CORI uses a Tf-Idf representation for each document cluster making possible to calculate scores for each document collection when a new query arrived. When web queries are conjunctive each collection score corresponds to the product of the partial scores obtained by each term in the collection shared with the query. Kulkarni and Callan (2010) proposed to organize large collections using the K-means clustering algorithm. The authors evaluated three document allocation strategies: random, source-based and topic-based, using document terms for content description (source and topic-based strategies). They showed that the vast majority of the relevant documents for CORI query routing are allocated in a small number of partitions. They also compared the search accuracy of exhaustive search with that of selective search. Results illustrate that selective search using topic-based partitions improves the search cost by an order of magnitude, emphasizing how useful collection selection is in vertical search.

Xu and Croft (1999) represented each collection using language models. In this way, given a query, each collection obtains a score that represents how likely it is for query terms to be observed in each collection. As GLOSS and CORI systems, Xu and Croft's strategy requires the use of the text from each document stored in each collection using a significant amount of space in the broker.

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

Kulkarni, Tigelaar, Hiemstra, and Callan (2012) proposed three algorithms in the context of topic-based partitions to decide which collection and how many partitions to select. It used the so called central sample index (CSI) as an inverted index of a small sample of randomly selected documents from each partition. The document ranking provided by CSI is transformed into a hierarchical tree which is traversal in a bottom up way to obtain the partitions ranking and to estimate the minimal rank cutoff.

Cambazoglu, Varol, Kayaaslan, Aykanat, and Baeza-Yates (2010) presented an approach for query routing in the context of Web search engines distributed geographically. The proposal used training queries to infer the upper bounds on document scores for a given query at each of the non-local indexes. These bounds are then used to predict if a non-local index would contribute with any document to the top-k results for the query which in turn determines if the query is forwarded to that non-local index. Results are further improved by using caching and replication.

Puppin, Silvestri, Perego, and Baeza-Yates (2007) proposed a method where a query log is used to construct clusters of documents. Puppin et al. used the algorithm introduced by Dhillon et al. (2003) to co-cluster documents and queries which frequently co-occur in query logs. Using a scalable implementation of co-clustering Puppin, Silvestri, Perego, and Baeza-Yates (2010) constructed a solution for queries and documents over a distributed system with 16 processors. Maintaining the queries text for each cluster, they calculated BM25 scores which measure how relevant each cluster is to a new query. This collection selection method, known as PCAP, has the advantage of using less space in the broker than its predecessors due to the fact that each cluster is represented using the query term space, leading to sparse text representations. Experimental results show that PCAP achieves better results than GLOSS and CORI over document partitioned collections, reason why it is considered as the state of the art for this problem.

3. Background

In this section, we describe the fundamental aspects that must be considered when building the vertically partitioned search engine, in the context of which we propose our collection selection method. In Section 3.1, we describe the term and document co-clustering algorithm used to build the document partitions. In Section 3.2, we describe the collection selection method known as PCAP, currently the gold standard against which we have compared our proposal.

3.1. Information theoretic co-clustering

The term and document co-clustering algorithm described in this section was originally proposed by Dhillon et al. (2003) for dyadic data. It has been employed for text (dyadic in terms versus documents) by Puppin et al. (2010) and to construct a document partitioning scheme in a search engine. Puppin et al. used the co-clustering results in their collection selection method, as we show in Section 3.2; thus, their method generates a dependence on the particular document partition. For comparison and evaluation, we are building our search engine using the same document partitioning algorithm. However, we believe that our proposal is more flexible because it is independent of the document partitioning strategy.

We describe the co-clustering algorithm using Dhillon's notation. Let $P(X, Y)$ be a contingency matrix that describes the joint probability between two discrete random variables X and Y. A co-clustering algorithm maps X into k disjoint hard clusters and Y into l disjoint hard clusters. Thus, a co-clustering algorithm finds maps $\hat{X} = C_X$ and $\hat{Y} = C_Y$, where $\hat{X}$ and $\hat{Y}$ are discrete random variables that are deterministic functions of X and Y, respectively. An optimal co-clustering minimizes the loss of mutual information between $P(X, Y)$ and the target co-clustering matrix $P(\hat{X}, \hat{Y})$.

Because C_X and C_Y define hard clustering maps, the target co-clustering matrix is defined by $P(\hat{X}, \hat{Y}) = \sum_{x \in \hat{X}} \sum_{y \in \hat{Y}} p(x, y)$. Defining $Q(X, Y) = P(\hat{X}, \hat{Y}) \cdot P(X \mid \hat{X}) \cdot P(Y \mid \hat{Y}) = P(\hat{X}, \hat{Y}) \cdot \frac{P(X)P(Y)}{P(\hat{X})P(\hat{Y})}$, we can express the mutual information loss function in terms of the Kullback-Leibler divergence:

$$D(P(X, Y) \mid\mid Q(X, Y)) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log \left(\frac{p(x, y)}{q(x, y)} \right).$$

Finding an optimal co-clustering solution is equivalent to minimizing $D(P(X, Y)||Q(X, Y))$ with the constraints for the number of clusters per row (k) and column (l). Dhillon et al. (2003) propose estimators for the functions $\hat{X} = C_X$ and $\hat{Y} = C_Y$ that can simultaneously maximize the mutual information between row clusters and column clusters. Further details on the algorithm are included in Dhillon et al. (2003).

3.2. PCAP

We describe PCAP following the notation provided by Puppin et al. (2010). Let Q be a collection of m distinct queries, and let D be a collection of n different documents. D can be retrieved from a black box search engine after executing the queries that belong to Q. Let $P(Q, D)$ be the contingency matrix that describes the joint probability between Q and D, where each entry $p(q_i, d_j)$ can be calculated from the score $r_{i,j}$ of document d_j in the list of the results of q_i.

Running a co-clustering algorithm on $P(Q, D)$, we obtain the target matrix $\hat{P}(\hat{Q}, \hat{D}) = \sum_{q \in \hat{Q}} \sum_{d \in \hat{D}} p(q, d)$. Puppin et al. (2010) employed each element of $\hat{P}(\hat{Q}, \hat{D})$ as an entry to the collection selection algorithm, which is referred to as PCAP, due to the symbol used to represent the estimator of the target matrix $\hat{P}$.

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

PCAP works in the following manner: For each query cluster in $\hat{P}(\hat{Q}, \hat{D})$, a dictionary is constructed with the terms that comprise the queries in each cluster. Whenever a new query is processed, the BM25 score between the query terms and the term dictionary of each query cluster is calculated in the broker. We represent the BM25 score obtained by query q in cluster qc_i with the variable $r_q(qc_i)$. Then, using the elements $\hat{p}(i, j)$ of matrix $\hat{P}(\hat{Q}, \hat{D})$, we are able to obtain the ranking $r_q(dc_j)$ of each document cluster dc_j using the expression

$$r_q(dc_j) = \sum_i r_q(qc_i) \cdot \hat{p}(i, j).$$

Then, the broker can route q to each document partition in decreasing order according to the scores $sr_q(dc_j)$. Further details and examples about how PCAP works are included in Puppin et al. (2010).

4. Learning-based collection selection

In this section, we explain how to build a training set to be used as input for the learning algorithm and capable of maximizing a reward function for relevant results given by the collection selection method. Our method begins by calculating the top-k document lists for each query submitted to the search engine, using a broadcast. As the queries are evaluated using a broadcast, the broker requests from each partition the top-k most relevant documents to the query. Then, the broker consolidates the lists by constructing a new list with the global top-k most relevant documents. This can be done due to scoring consistency across partitions. For this reason, these lists can be considered to be the gold standard of the learning process because they are constructed over the entire system. Note that this approach requires to execute a query against all the processors (multicast evaluation mode) at least one time previous to its inclusion in the query training set. Note also that a new multicast evaluation of the query is needed every time that the document collection is reorganized (for example, after each co-clustering rerun).

Because we employ a vertical partitioning scheme, each document is located in a single partition. Beginning from the global top-k document lists that correspond to each query, we proceed to elaborate a list of partitions that contain the global top-k documents. For each query $q \in Q$, we have a partition list $L(q)$ that enables a recall of the top-k best system wide results.

The partition lists $L(q)$ can be sorted according to a reward function for relevant results, in which the first partitions of each list contribute with the most relevant results to each query. This sorting enables us to route the queries to each document partition in decreasing order according to the reward function, which is a key element for the collection selection method design. A natural method for evaluating the reward of each partition consists of calculating the fraction of relevant documents that contributes to the global top-k list; this fraction corresponds to the partition's recall. Another reward function that we consider is the NDCG score which is added to the top-k list by each partition. The NDCG score also considers the position in which the documents appear (in the top-k list), this yields a higher score for the relevant documents that appear near the top of the list.

From the partition lists $L(q)$, we construct a training set for the learning algorithm that considers a given reward function. The main idea in our method is building the training set over the vector space of terms in Q using the reward function of the corresponding partition as a weight function for each given term. Each partition is represented over the space in terms of Q, which favors the evaluation of new queries that share terms with the vocabulary of Q.

We propose the learning problem in a supervised environment. Our method considers each partition as a class; therefore, the learning problem consists of finding the hyper-planes that separate the classes in term space. Each query is going to be modelled as a training instance; the query will contribute to the description of each partition (class), considering the terms it is composed of and weighting them according to the reward function associated to the corresponding partition. This is a multi-class scenario as we consider search engines with more than two partitions. In addition, it is a multi-label classification scenario because each list $L(q)$ can consider several partitions.

4.1. Set-up of a training instance

Our method transforms each query $q \in Q$ into a training instance, using its partition list $L(q)$. For each partition in $L(q)$ we create a training instance associated with q as follows: Let dc_j be a document partition in $L(q)$, and let t_i be a term that is associated with q. A vector representation of q is constructed, labeled dc_j and referred to as q^j over the vocabulary of Q. There are nonzero elements for the terms t_i that comprise q, which is given by the weight function w in which each nonzero component of q^j is given by $q^j(i) = w(q, dc_j)$. For the remaining vocabulary components not considered in q, the weight function returns a value of zero. This process constructs a sparse representation of q.

Several weight functions can be defined. We propose to use one of the following three functions:

- **Boolean function**: This function is the simplest of the three functions. It considers whether a given partition in $L(q)$ contributes to the top-k list. Thus, the weight function can be expressed as

$$w(q, dc_j) = \begin{cases} 1, & \text{if } dc_j \text{ retrieves at least one relevant document for } q \\ 0, & \text{otherwise.} \end{cases}$$

- **Recall function**: This function considers the global top-k document list as the gold standard. The weight function can be expressed as

$$w(q, dc_j) = \texttt{RECALL}(dc_j, q),$$

that is, the fraction of documents relevant to q retrieved from partition dc_j.

- **NDCG function**: We employ the definition of the discounted gain function $DG(i)$ used by Järvelin and Kekäläinen (2002) for a given position $i \geq 2$ of the global top-k document list, which is defined as $\frac{rel_i}{\log_2(i)}$, where rel_i represents the relevance of the document in the global top-k list. In the case of the list's first position, $DG(1)$ is simply defined as rel_1. In a list with k documents sorted by relevance in descending order, a decreasing integer function is typically used, with the initial value k (for the most relevant document) and the final value 1 (for the least relevant document). In this case, the discounted gain of the ideal list can be expressed as $DG(i) = \frac{k-i+1}{\log_2(i)}, i \geq 2$.

Example. Let us assume that the top-10 documents are considered for each query. Then, the ideal list (which sorts the documents by relevance in descending order) obtains the following discounted gains:

$$DG(1) = 10, DG(2) = 9, DG(3) = \frac{8}{\log_2(3)} = 5.06, \ldots, DG(10) = \frac{1}{\log_2(10)} = 0.301.$$

Let us consider the DG reward from the single partition dc_j. We need to compare the list of relevant documents retrieved from each partition with the global top-k list. Let us assume that the partition dc_j contains $1 \geq n \geq k$ relevant documents from the global top-k list. In the best-case scenario, the n retrieved documents correspond to the top-k global documents list; in this case, the partition gain returns a maximum value. The worst-case scenario corresponds to the partition dc_j, which retrieves only one relevant document from the top-k list, and the document is located in the last position of the list. We obtain the discounted gain fraction that is returned by partition dc_j by accumulating the DG obtained for each document (cumulative) and normalizing the DG with respect to the DG of the ideal list (normalized) as follows:

$$NDCG(q, dc_j) = \frac{\sum_{d_i \in dc_j \cap top-k} DG(i)}{\sum_{d_i \in ideal} DG(i)}.$$

Example. Let us assume that the global top-10 documents are considered for each query. We assume that partition dc_j returns three documents -d_3, d_5 and d_7- of the global top-k list. The ideal list that could be generated by a given partition (truncated to three results) corresponds to the documents d_1, d_2 and d_3. Then, the normalized gain returned by partition dc_j is calculated as

$$NDCG(q, dc_j) = \frac{8 + 6 + \frac{4}{\log_2(3)}}{10 + 9 + 5.06} = 0.68.$$

In the event that a partition has a null intersection with the global top-k list, $NDCG(q, dc_j) = 0$. We can easily verify that the maximum value of $NDCG$ is achieved when the partition returns the first documents of the global top-k list, in which case the $NDCG$ value is 1.

When using $w(q, dc_j) = NDCG(q, dc_j)$, the vector representation of each query is modeled by the discounted gain reward function over the vector space of the terms that correspond to each query.

4.2. Learning algorithm

The supervised learning process corresponds to a multi-class, multi-label scenario, which indicates that the learning algorithm must be capable of building several hyper-planes in the query term space (one for each partition of the search engine).

Given the query q and the specific partition dc_j, we build the vector representation of q that is associated to dc_j using a weight function $w(q, dc_j)$ over the terms of q, which we denote as q^j. The learning process is capable of determining a separating hyper-plane in query term space that is associated with the partition dc_j, which we represent with the variable w_j. The negative likelihood function, which corresponds to the information-loss function between the queries whose lists include the partition and the model w_j, is minimized. Let $Q^j = \{q_1^j, \ldots, q_l^j\}$ be the set of queries whose partition lists include dc_j. The information-loss function is given by

$$\inf - \text{loss}(dc_j, w_j) = C \cdot \sum_{i=1}^{l} \log(1 + e^{-y_i \cdot (w_j \cdot \vec{q_i} + w_0)}) + \frac{1}{2} w_j^t \cdot w_j,$$

where $y_i = 1$ if $q_i \in Q^j$ and $y_i = -1$ if $q_i \notin Q^j$. The variable $C > 0$ corresponds to the penalty factor, and the term $\frac{1}{2} w_j^t \cdot w_j$ corresponds to the L_2 regularization factor of the loss function. The penalty factor controls the balance between loss and

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

regularization, which prevents overfitting of $Q^j = \{q_1^j, \ldots, q_l^j\}$ by model w_i and the loss of generalization capabilities for new queries.

Given the new query q^{new}, which can be represented in the term space of Q, we can estimate the conditional generating probability of partition dc_j from q^{new}, using the logistic function over w_j as follows:

$$P(y = +1 \mid q^{\text{new}}, w_j) = \frac{1}{1 + e^{-(w_j \cdot q^{\text{new}} + w_0)}}.$$

The decision threshold corresponds to 0.5 because the estimation of $P(y = +1 \mid q^{\text{new}}, w_j)$ transforms the decision problem into a binary problem given the partition dc_j. Therefore, if $P(y = +1 \mid q^{\text{new}}, w_j) > 0.5$, we can add dc_j to the partition list $L(q^{\text{new}})$.

Because the learning of each hyper-plane for each partition is reduced to a binary problem, a logistic regression algorithm can serve as the learning algorithm. We use a logistic regression implementation that is designed to work with high-dimensional data and is referred to as liblinear, version 2.01.[1] Liblinear utilizes a logistic regression implementation based on a Newton method for trust regions. Further details of the logistic regression algorithm and its implementation in Liblinear are included in Fan, Chang, Hsieh, Wang, and Lin (2008).

5. Incremental learning strategy for collection selection

To add new documents to the collection we follow a simple strategy. Because we have a description of each partition over the query term space, adding a new document to the partition that has the greatest cosine similarity to the vector representation of the document is sufficient. This addition prevents the need to recalculate the co-clusters and rebuild all partitions with the associated cost of moving the documents between partitions. We will show in our experiments that this method avoids co-clustering recalculation with a small cost in terms of precision.

New queries can also be submitted to the search engine. This situation is common; thus, we must determine which aspects of our proposal must be reconsidered in this scenario. One point is related to the predictability of the frequency of a query. The logic of this point is that a query that is not expected to be frequent in the future does not need to be incorporated in the training data set. Several studies address this particular issue (Fagni, Perego, Silvestri, & Orlando, 2006; Gan & Suel, 2009; Lempel & Moran, 2003; Long & Suel, 2005) and apply query-caching strategies with admission/eviction policies. This approach enables the use of a reduced query set, with the goal of efficiently administrating the assigned cache memory. This aspect is independent of the learning model, which assumes as a previous condition that the queries will not marginally occur. Notice that both components may independently and sequentially operate without incorporating structural coupling. For this reason, we assume that our routing model can be coupled to a standard query-caching system with proper logic that is independent of the learning algorithm employed by the router. Therefore, it is beyond the scope of this study.

One aspect that is relevant to the learning algorithm is related to the detection of novelty in a query. The logic behind our learning algorithm is that any query that introduces novelty to the model must be added to the training data set. Therefore, the main element that must be modeled is novelty detection. We model novelty detection using a listwise-based detection strategy.

Let q^{new} be a frequent query that is not considered in the training data set, and let $L_M(q^{new})$ and $L(q^{new})$ be the partition lists that are obtained using the collection selection method and the complete system in broadcast, respectively. We evaluate the novelty of a query for the model as a function of the number of inversions between partitions and between the exact list $L(q^{new})$ and the approximate list $L_M(q^{new})$. The logic of this design criterion is that a low number of inversions indicates that both lists were produced by the same model. On the other hand, a high number of inversions in the partitions indicates that the model list is not a suitable approximation of the exact list; therefore, it should be added to the training data set.

Let $R_M(P)$ and $R(P)$ be the rankings of partition P in $L_M(q^{new})$ and $L(q^{new})$, respectively. For any pair of partitions P_1 and P_2, if $R(P_1) > R(P_2)$, we count a match if $R_M(P_1) > R_M(P_2)$; in the opposite case, we count an inversion. Analogously, we count matches or inversions in the case when $R(P_1) < R(P_2)$ for the condition $R_M(P_1) < R_M(P_2)$.

Let n be the total number of partitions to be compared between both lists. The maximum number of inversions corresponds to $\frac{n \cdot (n-1)}{2}$, which occurs when $L_M(q^{new})$ is the reverse of $L(q^{new})$. Let A be the number of matches and let B be the number of inversions between both lists. If both lists were generated independently, it is expected that A and B were jointly produced by a random process. Then, to check the independence hypothesis we use a test designed to measure ordinal association between two random variables. The following statistics exhibit a normal distribution when A and B are jointly produced by a random process (null hypothesis)

$$z = \frac{3 \cdot (A - B)}{\sqrt{\frac{n \cdot (n-1) \cdot (2n+5)}{2}}}.$$

If $2 \cdot P(-z) < p$, the null hypothesis can be rejected with a significant p-value. As a rule of thumb, we must recall, for example, that $z \geq 2.3$ is significant at 0.05. If the null hypothesis cannot be rejected, we must incorporate the query into the training data set. Further details on measures of ordinal association are provided in Kruskal (1958).

[1] https://www.csie.ntu.edu.tw/~cjlin/liblinear/

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

After we have stored new queries, we can add them in batch to the data set to re-train the model. This process can be periodically performed, as we show in the section of experiments, without affecting the performance of the search engine. The training can be performed off-line; once it is completed, a user can proceed to replace the old model.

Note that our model is a long memory model, that is, it does not consider the elimination of queries from the training data set because queries that were previously frequent may become frequent in the future. Among the factors that explain this phenomenon is seasonality.

6. Experimental results

In this section, we evaluate the performance of the proposed method. After introducing the data set and the experimental setting in Section 6.1, we start discussing the results of parameter tuning and parametric sensitivity in Section 6.2. In Section 6.3 we discuss the costs of our proposal in terms of time and space required. Then in Section 6.4 we compare the performance in batch learning mode of the collection selection method with PCAP by varying the number of partitions. Section 6.5 evaluates index update and its impact in terms of precision. Then, in Section 6.6 we evaluate the proposed selective learning strategy. Finally, we conclude in Section 6.7 discussing practical implications of our proposal to a production system.

6.1. Data set

The data presented in this paper's experimental section were generated using the Yahoo BOSS search engine. The Yahoo BOSS API enables the same results that the production Yahoo search engine presents to users. To generate the document collection, we employed a Yahoo log that corresponds to a three-months period in 2009; this log contains 2,109,198 distinct queries that correspond to 3,991,719 queries. The query term space spans a vocabulary of 239,264 different terms. For each query considered in the log, we obtained the top 50 results that were retrieved using Yahoo BOSS, and built a collection of 50,864,625 documents with them.

To create the partitions, we separated the queries into two different sets using the same proportions evaluated by Puppin et al. (2010). We randomly selected 1,629,113 queries to build the training set, and reserved the remaining 602,862 queries for evaluation. From these evaluation queries, a subset of 8061 queries that do not intersect the documents retrieved by the training set were eliminated. Therefore, the definitive testing set was reduced to 594,801 queries.

The co-clustering algorithm by Dhillon et al. (2003) was run over the set of training queries; their respective top 50 document lists were obtained. The same number of query partitions evaluated by Puppin et al. (2010) was employed, that is, 128 query partitions, as well as a variable number of document partitions. Five distinct configurations were considered: 16, 32, 64, 128 and 256 document partitions.

For each query, lists of the partitions that allow retrieval of the top 20 best results for the entire system were compiled for both the training sets and the testing sets. This process was performed for the five systems configurations. These lists will enable us to evaluate the performance of the proposed method.

We evaluated the performance in terms of Intersection measurement, which is employed by Puppin et al. (2010) and defined as follows:

- **Intersection**: Let's call G_q^N the top-N results returned for a query q by a black box search engine and H_q^N the top-N results returned for q by our method. The intersection at N, $INTER_N(q)$, is the fraction of results retrieved by the proposed method that appears among the top-N documents in the black box search engine:

$$INTER_N(q) = \frac{|H_q^N \cap G_q^N|}{|G_q^N|}.$$

The $INTER_N$ measure for a testing query set is obtained as the average of the $INTER_N(q)$ measured for each testing query.

6.2. Tuning and parametric sensitivity

In this section, we explore the parametric sensitivity of the learning algorithm to the training data. All results correspond to the representation based on NDCG. No significant differences in terms of sensitivity were observed in the two remaining representations (Boolean and RECALL); for this reason, we have not included them here.

The first parameter that needs to be explored is the regularizing parameter. This parameter is represented by the variable C and controls the balance between the regularization and the regressors to prevent training data overfitting. Tuning this parameter using cross-validation of the training data set over 5 folds is common. For all configurations, the parameter value was successfully determined to be 0.01, independent of the number of partitions. This value is less than the default value, which is set to 1. Using a penalty that is less than the default value gives higher relevance to the regularizer, which indicates its importance in this problem.

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

Table 1
Sensibility analysis of the tolerance parameter e in terms of training time and model size for different values of P.

		16	32	64	128	256
	nnz	7,364,056	9,305,958	10,934,415	14,174,148	18,029,122
Time	e = 0.01	1h 31m 32s	3h 1m	8h 41m 6s	22h 19m 48s	3d 4h 26m
	e = 0.1	42m 42s	2h 5m 12s	6h 29m 54s	18h 35m 11s	3d 2h 6m
	e = 1	7m 48s	28m 36s	2h 27m 30s	9h 22m	1d 14h 44m
	e = 10	2m 24s	13m 30s	51m 18s	3h 19m 42s	11h 22m 36s
Model Size	e = 0.01	137 MB	274 MB	549 MB	1.1 GB	2.2 GB
	e = 0.1	138 MB	275 MB	550 MB	1.1 GB	2.2 GB
	e = 1	147 MB	291 MB	568 MB	1.2 GB	2.2 GB
	e = 10	137 MB	294 MB	597 MB	1.2 GB	2.3 GB

Table 2
Co-clustering computational times and model sizes for different number of processors.

PCAP	16	32	64	128	256
Co-clustering runing time	1h 13m 2s	1h 2m 27s	1h 13m 12s	2h 41m 53s	7h 54m 24s
Co-clustering model size	121 MB	134 MB	144 MB	154 MB	176 MB

The solver is an iterative algorithm, whose cost per iteration is $O(nnz \cdot I)$, where nnz is the number of nonzero entries of the training data set and I is the number of internal iterations that are required for convergence. The number of internal iterations I is proportional to an early termination parameter (stopping criteria) -the tolerance e- which corresponds to the difference between the information loss between two successive iterations. A difference less than e triggers early termination. Thus, the inverse of e is proportional to the numerical accuracy of the model. We will show that the numerical accuracy of the model (the numerical precision used to estimate each model parameter) for values of e less than 0.1 does not affect the performance of our routing method.

As the number of partitions increases, the number of nonzero entries nnz must also increase because the length of the partition lists increases when the system considers more document partitions. This relation can be observed in Table 1.

We can observe that nnz sublinearly increases with P because the lists are calculated over the global top-k document, with fixed k (20). Because the solver has a cost of $O(nnz \cdot I)$, the increase in the parameter P produces an increase in the training time. We evaluated the sensitivity of the method to various values of e. Table 1 shows the time that is required for the training phase and the size of the corresponding model. We observed that low tolerance values produce longer training times; the model size is independent of e but proportional to P. The relation between the model size and P is linear.

We evaluated the parametric sensitivity in terms of the Intersection measurement for different values of the parameter e. We presented the results for $INTER_5\%$, $INTER_{10}\%$ and $INTER_{20}\%$ over the testing data. We show these results in Fig. 1.

As shown by these results, the performance has a low sensitivity to e. The impact of a lower tolerance value does not represent a significant performance increase, which shows that the method converges to a suboptimal situation, in which the parameter e serves a marginal role. The low sensitivity enables us to use a value of e that does not generate excessive computing times. For this reason, we use the default value of 0.1 that is considered for e in the solver.

6.3. Learning costs

We compare the costs in terms of computational time and memory to the ones involved in PCAP. As a first step, PCAP needs to run a co-clustering algorithm. We use the codes provided in Puppin and Silvestri (2007), which are implemented in C++. Then, using query-document coclusters, PCAP builds a collection of query dictionaries, including the terms of the whole collection of queries that belongs to each partition. Query dictionaries get 31 MB of space in 6.5 s, for every number of processors used in these experiments. Computational times and sizes for co-clustering are reported in Table 2.

As Table 2 shows more processors implies an increase in model size. In addition the time involved in co-clustering for 128 and 256 processors is greater than for 16, 32 and 64 processors. However no significant differences were registered between computational times involved in co-clustering for 16, 32 and 64 processors illustrating that for a small value of P the convergence of the algorithm is independent of P.

Instead of building a query dictionary for each partition using co-clusters, our learning-based approach processes the log to build a training set. Then, we run logistic regression for each training set. Costs in computational time and space involved are reported in Table 3. We used the NDCG variant for these experiments, with e tuned to 0.1.

As Table 3 shows, the creation of the training dataset involves computational times in the order of seconds. As we create a training instance for each query q in the log and each processor that retrieve at least one relevant document for q, the size of the training dataset increases with P, ranging from 213 MB to 850 MB. As the cost per iteration of logistic regression is $O(nnz \cdot I)$, where nnz is the number of nonzero entries of the training data set and I is the number of internal iterations

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

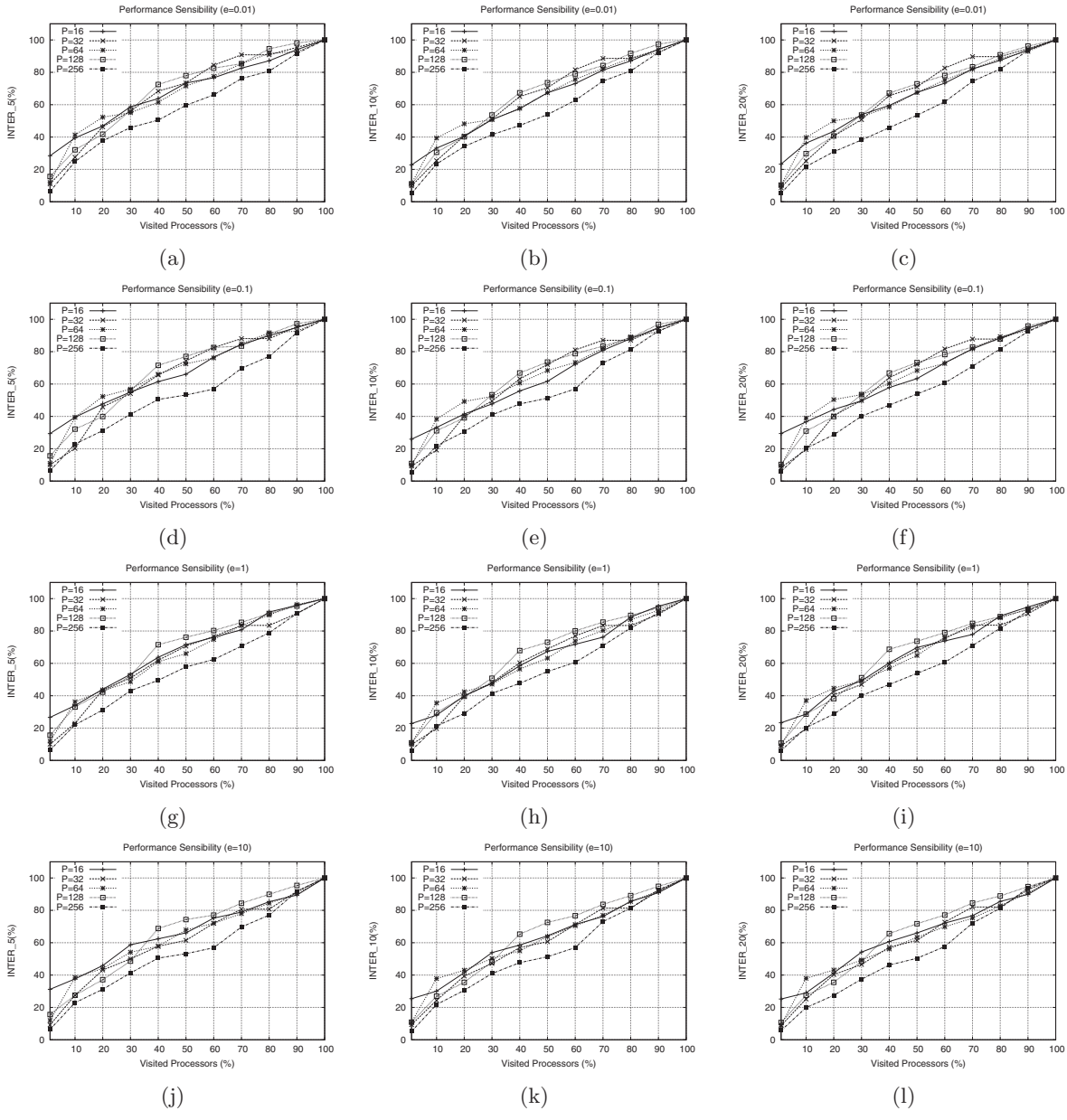

Fig. 1. Performance sensibility of the tolerance parameter e in terms of $INTER_5\%$, $INTER_{10}\%$ and $INTER_{20}\%$ measures for different values of P.

that are required for convergence, an increase in training set size implies an increase in computational time, as the third row of Table 3 shows. The last row of Table 3 shows model sizes, ranging from the order of MB to GB.

Comparing Table 2 and 3 we conclude that the cost of the learning approach is higher than the cost of PCAP. However, we claim that the costs of the learning approach reported in these experiments were obtained using a single core implementation of logistic regression. It is possible to speed up the learning step using a multicore implementation of logistic regression, as the one provided by Spark.[2] We suggest that the use of a multicore implementation is advisable and even indispensable in a large-scale production system.

[2] http://spark.apache.org

Table 3
Learning computational times and model sizes for different number of processors.

Learning strategy	16	32	64	128	256
Training dataset creation	49s	56s	1m 4s	1m 22s	2m 7s
Training dataset size	213 MB	271 MB	326 MB	501 MB	850 MB
Learning step	42m 42s	2h 5m 12s	6h 29m 54s	18h 35m 11s	3d 2h 6m
Learning model size	138 MB	275 MB	550 MB	1.1 GB	2.2 GB

Table 4
Batch learning evaluation using $INTER_5\%$, $INTER_{10}\%$ and $INTER_{20}\%$ for differents values of P. Results are shown for three configurations: (a) Visiting the top-1 recommended processor, (b) Visiting the top-25% of the processors, and (c) Visiting the top-50% of the processors. Bold fonts indicate best results for each configuration.

		$INTER_5\%$			$INTER_{10}\%$			$INTER_{20}\%$		
		Top-1	25%	50%	Top-1	25%	50%	Top-1	25%	50%
16	Log Boolean	**28.44**	**51.38**	73.39	**22.80**	**43.52**	67.36	**23.28**	**45.80**	**67.56**
	Log NDCG	22.94	44.04	74.31	20.21	41.45	**67.88**	21.37	42.37	65.65
	Log Recall	22.94	46.79	**75.23**	20.21	42.49	66.84	21.37	43.51	64.89
	PCAP	23.85	46.79	59.63	16.58	39.90	54.92	14.12	37.40	50.76
32	Log Boolean	11.01	**50.46**	70.64	9.84	**44.56**	**66.84**	8.78	**44.66**	**67.94**
	Log NDCG	**12.84**	42.20	**71.56**	10.88	40.41	66.32	11.07	42.75	67.18
	Log Recall	11.93	45.87	68.81	**11.40**	43.01	65.28	**11.45**	**44.66**	66.03
	PCAP	8.26	39.45	66.06	5.70	33.75	66.32	5.34	37.40	67.56
64	Log Boolean	**11.93**	**54.13**	**71.56**	**11.40**	**49.22**	67.36	**10.31**	**50.76**	67.56
	Log NDCG	7.34	48.62	**71.56**	7.25	48.19	70.98	6.87	47.71	71.76
	Log Recall	7.34	48.62	70.64	7.25	47.67	70.98	6.87	47.71	72.14
	PCAP	8.26	44.95	68.81	6.22	47.15	**72.54**	7.63	46.95	**74.81**
128	Log Boolean	**15.60**	**52.29**	**77.98**	**10.88**	**50.26**	**73.58**	**10.31**	**50.38**	**72.90**
	Log NDCG	14.68	**52.29**	73.39	10.36	49.22	70.47	9.92	48.85	72.14
	Log Recall	14.68	50.46	72.48	10.36	48.19	69.43	9.92	48.85	70.99
	PCAP	8.26	34.86	66.97	5.18	33.68	61.66	5.73	34.35	61.83
256	Log Boolean	6.42	**41.28**	58.72	5.18	37.82	53.89	5.34	35.50	53.05
	Log NDCG	**7.34**	**41.28**	**62.39**	**5.70**	**45.08**	**65.28**	**5.73**	44.27	**62.98**
	Log Recall	**7.34**	40.37	60.55	**5.70**	**45.08**	63.21	**5.73**	**44.66**	61.45
	PCAP	1.83	31.19	58.72	1.55	33.16	59.07	1.91	32.82	58.78

6.4. Batch-learning experiments

In this section we present the performance evaluation of our method and PCAP in terms of the indicators $INTER_5$, $INTER_{10}$ and $INTER_{20}$. To evaluate each method's ability to sort the partitions by relevance, we evaluate over the best partitions of each list for each configuration. For example, for the 16-partition configuration, we evaluate $INTER_5$, $INTER_{10}$ and $INTER_{20}$ for the first, fourth and eighth partitions of each list elaborated with the collection selection method. To compare the evaluations of the configurations with different partition numbers, we use the top 1, the top 25% and the top 50% for all evaluations.

The evaluation is performed over the test query data set. The results presented in the following table show the averages for each configuration for all test queries. The proposed method is indicated with the prefix Log (for **log**istical regression). The suffix indicates the weight function employed to build the training instances.

The columns in Table 4 show the number of partitions for each of the lists considered. For a given configuration, the performance improves as more partitions from each list are considered. The comparison of the performance of different configurations indicates that in general it decays as more partitions are considered in the system. For example, the value of $INTER_5\%$ over the first partition for Log Boolean decreases from 28.44% in the 16-partition configuration to 6.42% when 256 partitions are considered. However, there are some configurations where the metric peak is achieved for 128 processors. For example, Log Boolean reaches its best result visiting the 50% of the processors at 128. Regarding the use of the indicators $INTER_5\%$, $INTER_{10}\%$ and $INTER_{20}\%$, the increase in the value of N from 5 to 20 produces a more restrictive evaluation, which is reflected in the fact that the $INTER_{20}\%$ results are slightly less than the results obtained using $INTER_{10}\%$; the $INTER_{10}\%$ results are less than the $INTER_5\%$ results. In some configurations, these differences are more significant.

The performance always surpasses the baseline given by a uniform, random assignment method. In the case of 256 partitions, a uniform random assignment method would achieve a performance of $\frac{1}{256}$ over the first partition, that is, a base of 0.39%. For this reason, a seemingly low performance of 6.42% over the first partition is, actually, not low because it surpasses the baseline by a factor of more than 16. For the same measurement, PCAP surpasses the baseline by a factor of 4.5 (achieving 1.83% in the first partition), which shows the significant improvement introduced by our method.

Discussion. As demonstrated by our results, our method surpasses PCAP in almost all the configurations for any of its three weight functions. The only exception is for 64 processors, where PCAP is very competitive in terms of $INTER_{10}\%$ and

$INTER_{20}$%. For 16 partitions, the best variant of our method corresponds to Log Boolean, which achieves a significant difference in the first partition. This difference decreases as more partitions are considered; it is overcome in $INTER_5$% by Log Recall and in $INTER_{10}$% by Log NDCG. For 32 partitions, the results are more uneven, with slight differences in favor of a particular function. In this configuration, the performance of Log Boolean decays at the first partition but reduces its disadvantage with respect to the remaining functions. This finding shows the positive effect of Log Recall or Log NDCG in fitting the documents that occupy the first positions of the list and the impact of the performance over evaluations that are truncated at the first partition. This advantage is lost in the configurations based on 64 partitions, where Log Boolean obtains the best results for the three measurements. A slight difference in favor of Log NDCG versus Log Recall is observed. For 128 partitions, the difference in favor of Log NDCG is distinct, whereas the difference in favor of Log Boolean is clear for 256 partitions. These results are not sufficient for establishing distinct differences in favor of one weight function; the only valid conclusion that can be drawn for all configurations is that our method consistently surpasses PCAP. Note that these results may be attributed to the fact that the collection is formed over global top-50 document lists, from which we have employed configurations with many partitions. If this restriction disappears we expect the effect in favor of NDCG becomes more significant. For these experimental settings, however, a rough reward function, such as a Boolean function, appears to be sufficient.

6.5. Index update

We start this section measuring the size of each partition and the quality of the results for different document index update strategies. We add new documents to the collection by calculating the cosine similarity of each new document to the vector representation of each partition in the query term space. Then, each document is assigned to the closest partition using its content as a query over the query routing model. This strategy avoids the recalculation of the query routing model, limiting the cost for index update at the level of each local index.

A crucial point to evaluate is the relative size of the partitions achieved using index update. As expected, an unbalanced partition size solution will increase latency.

We measured the size of the partitions for four different configurations: a) Full collection: We run co-clustering over the whole document collection, i.e. considering 50,864,125 documents, b) Partial collection: We run co-clustering over a fraction of the document collection which comprises 40,000,000 documents. Then we updated the collection by assigning the remainder 10,864,125 documents using as a routing method c) PCAP update: we use PCAP to distribute the documents, and d) Log NDCG update: we use Log NDCG for document distribution.

We show in Figs. 2 and 3 the size of the document collections in each partition for different values of P. The expected partition size m ($m = \frac{N}{P}$, N: total number of documents) is depicted with a segmented line. Note that a balanced solution will tend to be around m. To perform a fair comparison among the different configurations, all the plots shows normalized collection sizes (see the x axis), ranging from 0 to $2m$.

Figs. 2 and 3 shows that co-clustering achieves balanced solutions for P in 16 to 64. However, as P increases, solutions tend to be more unbalanced. In fact, for 128 and 256 processors, on full and partial collections of documents, both co-clustering solutions accomplish partition sizes beyond scale. This fact illustrates that co-clustering overemphasize the relevance of document and query pairwise distances, lacking of a density condition that may helps getting balanced cluster size solutions.

The third and fourth columns of Figs. 2 and 3 show the number of documents (normalized to get 10.8 M of docs) assigned to each partition using PCAP and Log NDCG, respectively. As we can see from the figures, PCAP gets unbalanced solutions. In fact, as P increases, PCAP tends to concentrate the documents into a single partition. These results illustrates that the use of document as queries over query dictionaries skews the results, phenomenon that is more noticeable for 128 and 256 processors. In fact, under these configurations the 89% and 98% of the documents were routed to a single partition. On the other hand, Log NDCG achieves more balanced solutions. The effect of partition unbalance is more noticeable for 128 and 256 partitions, where a few processors concentrate the half of the documents. However, the remaining partitions capture documents achieving sizes around m. This fact may be explained from the nature of the model. Logistic regression builds models with nonzero entries (see model sizes in Table 3) achieving a smoothed representation of each partition over the query term space. Accordingly, the use of long term vectors as the ones created from document contents over a smoothed model will tend to limit the effect of bias, favoring partition balance.

Fig. 4 shows the impact of index update on the routing method in terms of precision. We use as a performance measure $INTER_{10}$%. Fig. 4 shows that both index update strategies obtain performances that are between the results achieved using partial and full collections. Note that the evaluation was conducted using testing queries whose results were retrieved from the whole collection (50 millions of documents), reason why the precision of partial collection only gets the 80% in terms of $INTER_{10}$% visiting all the processors. Accordingly, Log NDCG and PCAP over the partial collection are considered as baselines. Fig. 4 shows that PCAP update (depicted with a circle) is almost equal to the baseline. On the other hand, Log NDCG update (depicted with a bold square) is always several precision points over the baseline, reaching the performance of PCAP on the full collection. The effect of unbalance explains why Log NDCG update cannot reach the performance of Log NDCG on the full collection, suggesting that Log NDCG update is an acceptable solution for index update during a transient. However, to avoid precision depletion the system will need to recalculate co-clustering forcing document reassignment from time to

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

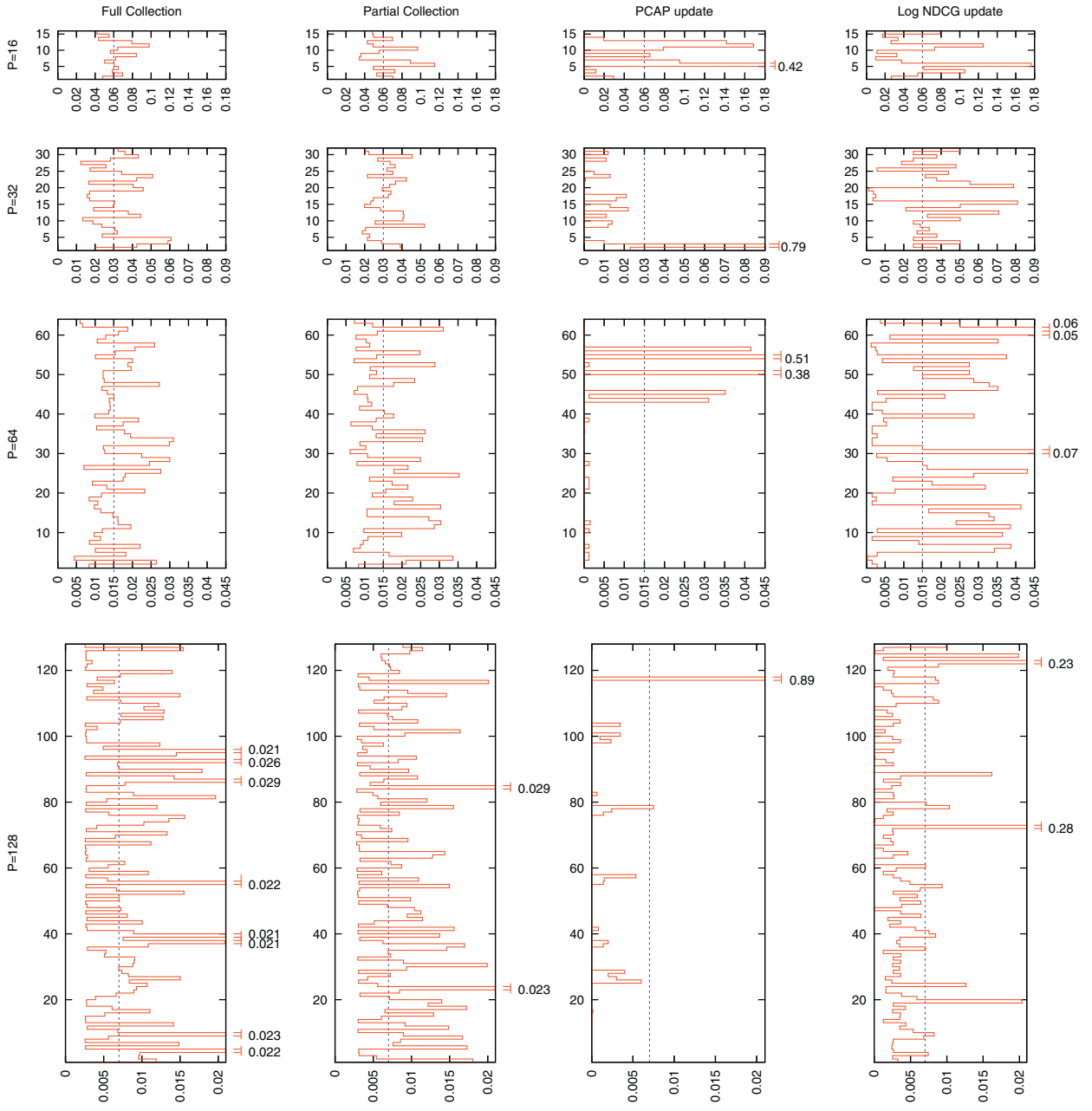

Fig. 2. Size of the document collections in each partition for P in 16 to 128. Each segmented line indicates the expected partition size for perfect balance ($m = \frac{N}{P}$, where N is the size of the collection). The x axis shows normalized collection sizes ranging from 0 to 2 m. Out of range partition sizes are indicated showing the size of the respective partition at the right side of each plot.

time. To avoid the inclusion of these new costs the use of an incremental clustering solution appears to be an advisable strategy to scale up our proposal to a production system.

6.6. Incremental learning experiments

To evaluate our incremental learning strategy, we have chronologically separated the test query set into three groups of similar sizes. The first part of the test set considers 187,204 queries; the second part of the test considers 192,400 queries; and the third part of the test set considers 215,197 queries. Each of the three sets spans timelines of approximately one month to make them comparable. For each third of the test set, we evaluate its performance using the $INTER_5\%$, $INTER_{10}\%$ and $INTER_{20}\%$ indicators considering three learning strategies: The first strategy does not incorporate any query from the

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

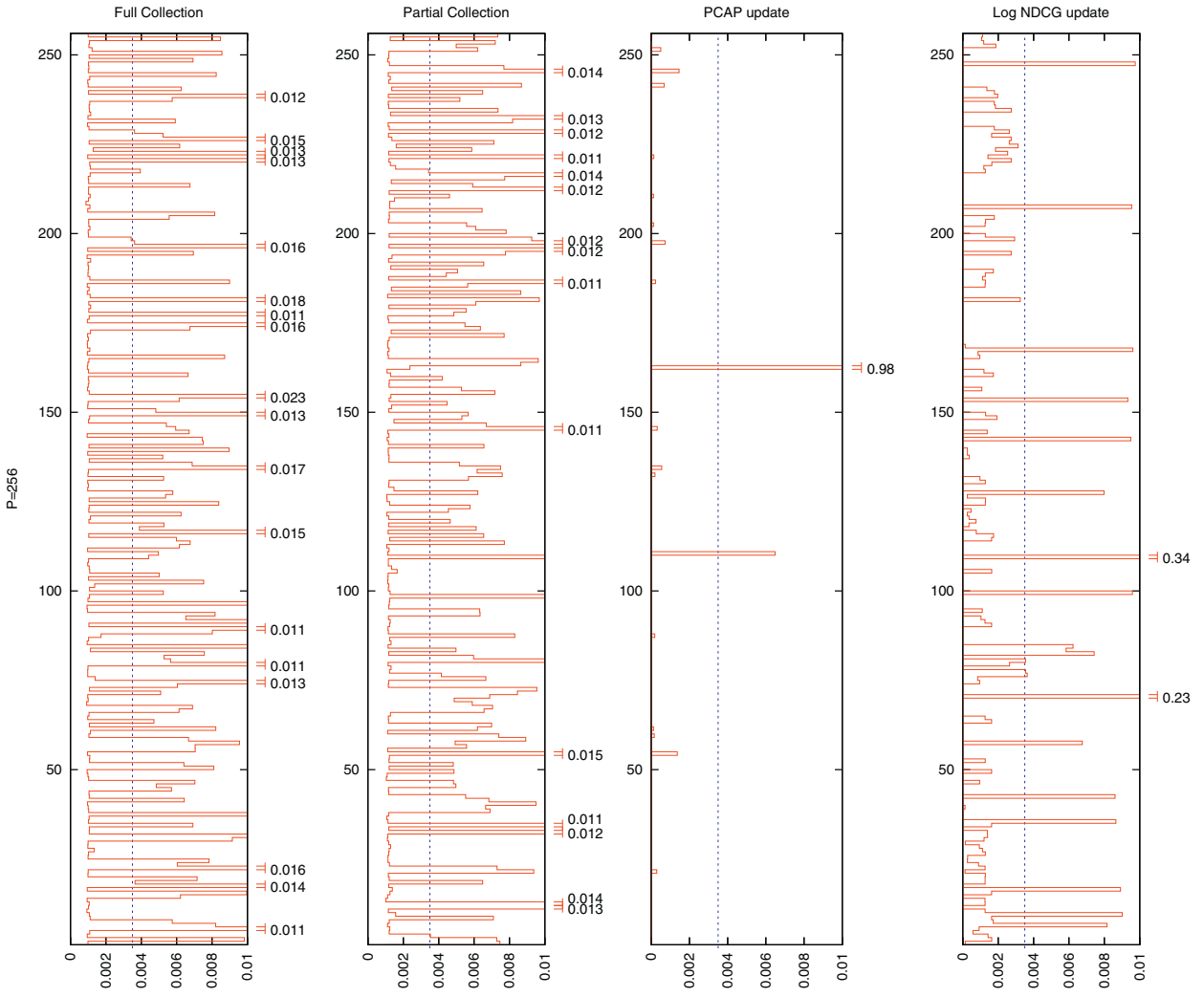

Fig. 3. Size of the document collections in each partition for 256 processors. As in Fig. 3, the expected partition size is indicated with a segmented line. Out of range partition sizes are indicated at the right side of each plot.

test set into the model. This strategy was evaluated in the previous section and was denoted batch learning. The second strategy, which we refer to as full learning, incorporates all queries into the model. The third strategy, which is denoted selective learning, selectively incorporates queries that satisfy the novelty criterion discussed above in Section 5.

Selective learning demonstrates important discrimination capabilities. When the query selection method is run with $z \geq 2.3$ that is, to a significance value of 0.05–32.07%, 31.43% and 34.08% of the queries from each third of the test query set are discarded. The novelty detection method discards almost a third of the queries to be incorporated into the training phase.

In the first evaluation, both incremental strategies performed the re-training at the end of the corresponding period. To consider the cumulative effect of the time invested in incremental learning in the batch case, each third of the timeline considers the queries from the previous period. Consequently, the evaluation over the final third of the timeline considers the full test data set; for this reason, this experimental configuration coincides with the batch evaluation that was performed in the section on batch learning.

The three strategies are evaluated for the four routing methods previously explored. Due to our query selection strategy is built over the partition lists, which corresponds to the output of a generic routing method, we can also apply it to PCAP. As measures of partition selection, for each search engine configuration, we consider the $INTER_5$%, $INTER_{10}$% and $INTER_{20}$% indicators, which are calculated over the first partition (top 1), top 25% and top 50% of the partitions with the highest routing score. In this evaluation, we consider three distinct search engine configurations with 16, 64 and 256 partitions. The results of this evaluation for $INTER_5$, $INTER_{10}$ and $INTER_{20}$ are listed in Table 5, 6 and 7, respectively.

The results displayed in Table 5, 6 and 7 show that the selective incremental learning strategy produces better results than its competition. In particular, selective learning surpasses full learning, which indicates that the training query selection

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

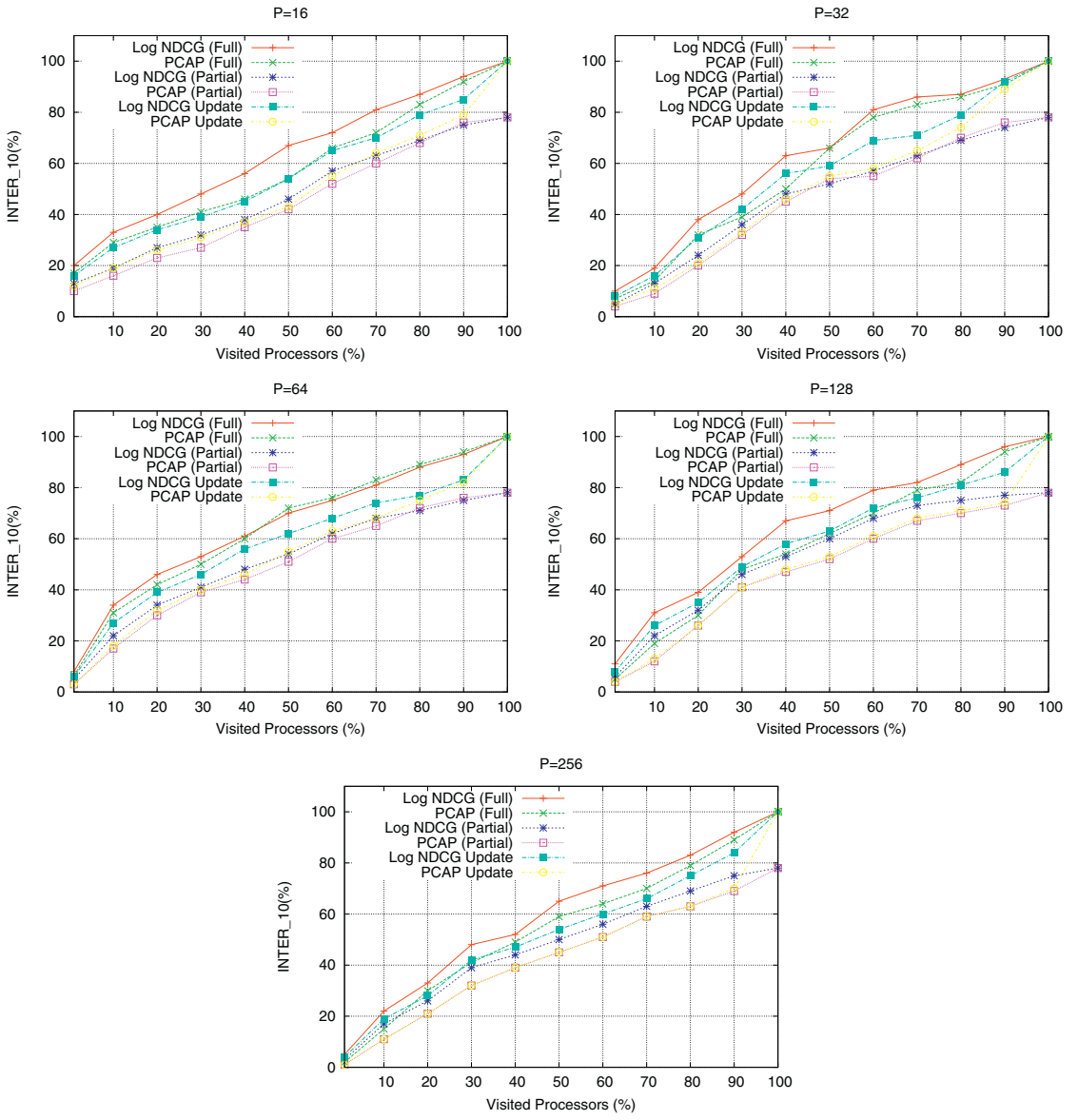

Fig. 4. Performance of routing methods in terms of $INTER_{10}\%$ for different index update strategies using P processors ranging from 16 to 256. Routing over the partial collection is a baseline for the experiment.

method enables the discarding of queries that are detrimental to the model's predictive performance. An explanation for this phenomenon is that when queries that can be approximated by the model are discarded, we prevent overfitting and improve the model's generalization capabilities. Both incremental learning strategies surpass batch learning, which confirms the need to periodically update the routing model so as to prevent a performance loss.

Discussion. For the $INTER_5\%$ indicator for 16 and 64 partitions, selective learning surpasses full learning in all comparisons. The best results are obtained by Log Boolean, closely followed by Log Recall. When the partition number is increased to 256, Log Recall and Log NDCG become more competitive and surpass Log Boolean in all comparisons. The fact that Log Boolean overcomes its closest competitors in configurations with 16 and 64 partitions indicates the presence of overfitting. When using the performance indicators $INTER_{10}\%$ and $INTER_{20}\%$ and considering the performance over more relevant documents, we increase the recall relevance to the performance. In this scenario, the learning algorithms that are based on recall functions (Log Recall and Log NDCG) overcome their competitors. For example, the use of $INTER_{10}\%$, Log NDCG with selective learning always disputes the first place in performance, for 16, 64 and 256 partitions. The difference becomes more significant in favor of Log NDCG for configurations with more partitions where differences between 7 and 8 percentile points in $INTER_{10}\%$ over Log Boolean are observed. This finding shows that Log NDCG has greater generalization capabilities as the recall increases. Using $INTER_{20}\%$, which represents the performance measure at the maximum reachable recall level

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

Table 5
Incremental learning evaluation using $INTER_5$%. Three different configurations were evaluated: (a) Batch learning, (b) Full incremental learning, and (c) Selective learning. Results achieved by visiting the top-1, top-25% and top-50% partitions are shown for each of three parts of the query log analyzed.

			First part			Second part			Third part		
			Top-1	25%	50%	Top-1	25%	50%	Top-1	25%	50%
16 partitions	Batch	Log Boolean	27.62	51.98	73.11	27.63	51.32	73.66	28.44	51.38	73.39
		Log NDCG	23.56	43.87	73.79	22.73	44.39	74.71	22.94	44.04	74.31
		Log Recall	23.63	46.32	74.46	23.05	47.03	75.08	22.94	46.79	75.23
		PCAP	23.79	47.48	58.78	23.93	47.08	59.13	23.85	46.79	59.63
	Full-inc	Log Boolean	33.71	58.78	79.94	33.73	58.23	79.87	32.64	58.01	79.88
		Log NDCG	29.58	50.29	80.28	29.68	51.31	81.03	29.01	50.94	80.77
		Log Recall	30.06	51.52	81.36	29.91	53.87	80.66	29.64	53.47	82.06
		PCAP	30.27	53.04	65.24	30.82	51.65	66.09	30.79	55.25	65.8
	Sel-inc	Log Boolean	**35.14**	**60.14**	81.81	**36.49**	**60.15**	82.63	35.01	**60.27**	82.55
		Log NDCG	30.97	52.71	83.57	31.08	55.82	83.41	31.85	55.81	86.29
		Log Recall	32.31	54.99	**85.95**	32.97	58.13	**86.99**	**35.38**	58.58	**89.97**
		PCAP	29.96	52.61	65.58	29.79	51.37	65.13	27.36	52.64	63.21
64 partitions	Batch	Log Boolean	11.08	54.95	71.72	11.53	54.29	71.06	11.93	54.13	71.56
		Log NDCG	7.79	47.65	71.94	7.13	48.09	71.47	7.34	48.62	71.56
		Log Recall	6.9	49.09	70.84	7.37	48.89	70.83	7.34	48.62	70.64
		PCAP	8.06	45.92	68.34	8.07	45.48	68.75	8.26	44.95	68.81
	Full-inc	Log Boolean	17.64	60.26	78.45	18.41	59.86	75.27	18.67	63.11	78.48
		Log NDCG	14.14	53.15	78.01	13.63	53.57	75.88	13.79	57.02	78.29
		Log Recall	13.83	54.73	77.65	14.15	54.13	75.15	13.69	56.76	76.89
		PCAP	14.86	51.04	75.31	14.94	51.03	73.51	14.41	53.84	74.89
	Sel-inc	Log Boolean	**19.32**	**62.18**	80.03	**21.03**	**61.76**	77.68	**21.55**	**66.02**	81.47
		Log NDCG	15.61	55.01	81.18	15.18	57.65	78.73	16.11	61.74	84.21
		Log Recall	15.81	57.84	**82.51**	17.29	58.31	**82.12**	19.51	62.35	**84.39**
		PCAP	14.38	51.18	75.02	13.19	50.08	73.42	9.99	51.18	72.01
256 partitions	Batch	Log Boolean	6.34	42.27	58.41	6.8	41.39	59.12	6.42	41.28	58.72
		Log NDCG	6.51	40.5	61.89	6.67	39.89	62.67	7.34	41.28	62.39
		Log Recall	7.91	41.02	60.14	7.01	40.93	60.38	7.34	40.37	60.55
		PCAP	1.53	31.93	58.38	1.63	31.45	59.04	1.83	31.19	58.72
	Full-inc	Log Boolean	12.95	47.27	65.35	12.95	47.16	64.06	12.99	49.91	65.56
		Log NDCG	13.09	46.27	67.93	12.76	45.19	67.35	14.25	49.45	69.33
		Log Recall	14.68	46.23	66.92	13.56	46.21	65.21	14.06	48.86	66.93
		PCAP	8.01	37.06	64.66	7.89	36.48	63.41	7.87	39.69	64.98
	Sel-inc	Log Boolean	14.33	48.27	67.23	15.46	49.12	66.01	15.12	52.14	67.51
		Log NDCG	14.76	48.78	**72.79**	15.95	49.73	**73.81**	**19.46**	**54.91**	**76.98**
		Log Recall	**16.99**	**48.92**	72.17	**17.22**	**50.81**	72.15	19.45	54.82	74.92
		PCAP	7.02	37.01	64.93	6.26	35.14	63.41	4.52	37.64	62.02

in these experiments, Log NDCG and Log Recall surpass Log Boolean in all comparisons. These differences decrease when 16 partitions are used. Note that PCAP is surpassed in all comparisons by the proposed methods.

The following experiment considers a more realistic evaluation scenario. In this experiment, the router is placed on-line with the stream of submitted queries and is updated on daily basis using the queries selected by the novelty detection method. The model is evaluated using the next-day test queries to avoid the intersection between the training query set and the testing query set. Note that in the case of 256 processors, logistic regression takes 3 days to converge in a single core implementation (see Table 3), reason why in a production system we will need a fast implementation of this algorithm. For the purposes of this experiment, we assume that the model may be obtained in at most 24 hours making possible the evaluation of the proposal for this scenario. Note that this constraint imposes the need to use a parallel implementation of logistic regression, that is beyond the scope of this paper.

For each day, performance values of previous evaluations can be obtained. In this evaluation, we consider the $INTER_{10}$% and $INTER_{20}$% indicators, over first-partition lists (top-1), and top-25% partitions. As in the previous experiment, we evaluate configurations with 16, 64 and 256 partitions. We compare selective learning over Log Boolean, Log Recall, Log NDCG and PCAP learning. Because the indicators are evaluated daily over the entire log, a time series is obtained over a timeline of 90 days. The results are shown in Figs. 5 and 6 for the $INTER_{10}$% indicators and $INTER_{20}$% indicators, respectively.

As shown in Figs. 5 and 6, the performance in terms of $INTER_{10}$% and $INTER_{20}$% shows mean values close to those observed in the batch evaluation, with reduced variability for the learning-based methods in $INTER_{10}$%, for 16, 64 and 256 partitions in the top-1 partition. All comparisons show advantages over PCAP, confirming the observations from the batch evaluation. The variability between the learning-based methods increases as more partitions are explored. The results over the top 25% for $INTER_{10}$% indicate a difference in favor of Log NDCG and Log Recall over Log Boolean, which increases for configurations with 64 and 256 partitions. When using the indicator $INTER_{20}$%, the three learning-based strategies obtain similar results for the top-1 partitions in the 16-partition configuration. Using 64, 128 and 256 partitions, Log NDCG and

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

Table 6
Incremental learning evaluation using $INTER_{10}$% for 16, 64 and 256 partitions.

			First part			Second part			Third part		
			Top-1	25%	50%	Top-1	25%	50%	Top-1	25%	50%
16 partitions	Batch	Log Boolean	22.22	42.64	68.14	22.79	43.41	67.72	22.80	43.52	67.36
		Log NDCG	19.45	41.82	66.43	19.9	41.64	67.14	20.21	41.45	67.88
		Log Recall	19.55	42.15	66.03	20.54	42.23	66.11	20.21	42.49	66.84
		PCAP	15.62	39.11	54.66	15.74	39.36	55.06	16.58	39.90	54.92
	Full-inc	Log Boolean	28.71	47.75	74.41	29.23	49.11	72.61	29.29	52.35	73.64
		Log NDCG	25.61	46.88	73.01	26.61	47.41	71.25	26.92	49.86	74.35
		Log Recall	26.46	47.27	72.91	27.04	47.72	70.48	26.93	51.38	72.86
		PCAP	22.32	44.81	61.44	22.59	44.81	59.44	22.71	48.12	61.81
	Sel-inc	Log Boolean	**30.02**	49.17	75.68	**32.18**	50.32	75.22	**32.16**	54.61	76.53
		Log NDCG	29.07	**51.04**	**80.31**	31.51	**54.17**	**79.55**	31.75	58.61	**84.22**
		Log Recall	28.59	50.79	78.99	30.01	52.67	77.08	29.93	**59.01**	81.71
		PCAP	21.89	44.66	61.26	20.88	44.12	58.89	18.79	45.29	59.66
64 partitions	Batch	Log Boolean	10.87	48.94	67.34	11.47	49.57	67.7	11.40	49.22	67.36
		Log NDCG	6.49	48.14	71.32	7.24	48.16	71.22	7.25	48.19	70.98
		Log Recall	6.72	47.64	70.86	7.17	48.58	70.88	7.25	47.67	70.98
		PCAP	6.13	47.74	71.88	6.27	46.78	72.03	6.22	47.15	72.54
	Full-inc	Log Boolean	17.83	53.99	73.54	18.31	55.23	72.29	18.32	58.14	73.94
		Log NDCG	12.78	53.16	78.17	13.81	53.52	75.94	14.17	56.87	77.68
		Log Recall	13.56	52.95	77.69	14.04	51.41	75.52	13.81	55.94	77.35
		PCAP	12.14	53.08	78.53	13.09	52.69	76.53	12.97	55.55	78.64
	Sel-inc	Log Boolean	**19.49**	55.02	74.89	**20.99**	56.78	75.28	**20.71**	60.33	76.24
		Log NDCG	14.41	**57.48**	84.04	15.98	**58.41**	**82.69**	17.17	**64.21**	**86.14**
		Log Recall	15.27	57.11	**84.11**	16.28	55.63	82.21	17.04	63.07	86.04
		PCAP	11.83	53.46	78.71	11.61	52.08	76.27	9.91	53.21	75.28
256 partitions	Batch	Log Boolean	5.36	37.6	53.39	4.45	37.91	54.34	5.18	37.82	53.89
		Log NDCG	5.66	44.51	65.07	6.01	44.55	65.81	5.70	45.08	65.28
		Log Recall	5.61	44.75	63.07	5.88	45.58	63.91	5.70	45.08	63.21
		PCAP	1.31	32.47	59.46	1.92	32.98	59.01	1.55	33.16	59.07
	Full-inc	Log Boolean	12.04	43.45	60.13	11.41	43.69	58.54	11.89	46.75	59.95
		Log NDCG	12.48	49.51	71.31	12.41	49.78	70.17	12.36	53.64	71.76
		Log Recall	11.75	50.48	69.46	12.01	50.97	68.89	11.99	53.66	69.46
		PCAP	7.51	37.72	65.48	8.59	38.05	63.16	7.83	41.44	65.77
	Sel-inc	Log Boolean	**13.72**	45.22	61.37	14.22	45.11	61.36	**14.15**	49.37	62.05
		Log NDCG	13.52	**52.33**	**76.09**	**14.33**	**54.05**	**75.74**	13.39	**59.55**	**79.56**
		Log Recall	12.29	51.56	71.26	13.74	52.98	72.09	13.26	57.41	73.52
		PCAP	7.22	37.43	65.14	7.14	37.54	62.66	4.05	39.42	63.37

Log Boolean establish a difference of approximately five performance points over Log Recall. Considering the results based on the top-25% partitions, the three learning-based strategies obtain similar results for 16 and 64 partitions, with a slight advantage in favor of Log NDCG. For the 256-partition configuration, the difference between Log NDCG and Log Recall over Log Boolean significantly increases, which reveals a gap of almost ten performance points.

Seasonal factors can be appreciated in the series for all evaluations. The day of the week is indicated on the horizontal axis, which reveals a decrease in performance by a few points during the weekend. In some comparisons, the seasonal factor can be highly relevant, as is the case of the evaluation over top-25% using $INTER_{10}$%. Using the $INTER_{20}$% indicator, the seasonal factor is less relevant indicating that seasonal relevant documents tend to appear in the top-10 page results. A slight improvement in learning can also be appreciated when using $INTER_{20}$%, where a positive slope is observed in the performance series for the strategies that are based on recall learning.

6.7. Practical implications to a production system

In this section we illustrate some implications of our proposal to a production system. We do this by evaluating efficiency aspects of our proposal in a parallel information retrieval system.

We evaluate our proposal in a BSP-based simulator Valiant (1990), distributing the cost relevant operations among the supersteps and processors, wherein the cost of each operation is determined by a set of process-oriented discrete-event simulation objects which model resource contention of operations. In each superstep, the use of these resources causes cumulative costs in the respective processors. The degree of contention is determined by the outcome of a cache admission/eviction policy. A first cache that is considered in the baseline strategy is at the level of each processor. As the length of the posting lists can be huge and during periods of time it usually happens that a large subset of terms is not required, most of the posting lists are kept in secondary memory. A subset of highly required posting lists is kept in main memory, what we called posting list caches. These caches are administered with a standard LRU admission/eviction policy in each processor. A second cache is maintained in the broker. This cache is a results cache which stores the answers for frequent

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

Table 7
Incremental learning evaluation using $INTER_{20}$%.

			First part			Second part			Third part		
			Top-1	25%	50%	Top-1	25%	50%	Top-1	25%	50%
16 partitions	Batch	Log Boolean	22.84	44.43	67.72	22.95	44.46	67.4	23.28	45.80	67.56
		Log NDCG	20.86	41.66	65.28	21.13	42.07	64.41	21.37	42.37	65.65
		Log Recall	20.68	42.58	64.86	21.41	43.37	65.04	21.37	43.51	64.89
		PCAP	13.43	38.33	50.73	14.16	37.54	50.91	14.12	37.40	50.76
	Full-inc	Log Boolean	29.51	49.49	73.51	29.28	49.66	73.88	28.49	51.89	73.59
		Log NDCG	26.93	47.41	70.93	27.73	47.84	71.28	26.84	48.44	72.45
		Log Recall	27.36	48.39	69.95	28.25	49.03	71.52	27.11	49.54	71.01
		PCAP	19.49	43.96	56.48	20.51	43.12	57.08	19.58	43.74	57.43
	Sel-inc	Log Boolean	**31.18**	51.46	**75.41**	**32.02**	50.71	**76.46**	**31.35**	**54.78**	76.12
		Log NDCG	29.05	50.53	75.17	29.74	51.7	76.19	30.23	53.35	**78.27**
		Log Recall	28.97	**51.75**	73.81	30.03	**51.81**	75.99	30.07	53.03	75.11
		PCAP	18.81	44.02	56.58	18.85	42.41	56.61	16.41	41.08	55.25
64 partitions	Batch	Log Boolean	11.16	50.65	66.57	10.86	51.28	66.79	10.31	50.76	67.56
		Log NDCG	6.72	48.16	71.87	6.5	47.79	70.98	6.87	47.71	71.76
		Log Recall	6.7	47.96	71.35	6.76	47.74	71.98	6.87	47.71	72.14
		PCAP	7.41	46.92	75.24	7.23	47.86	75.11	7.63	46.95	74.81
	Full-inc	Log Boolean	17.29	55.77	71.98	17.01	56.71	73.04	15.91	57.33	74.12
		Log NDCG	12.91	53.25	77.12	12.74	53.35	77.79	12.28	53.78	78.57
		Log Recall	12.91	53.31	76.62	13.43	53.71	78.65	12.81	54.51	79.08
		PCAP	13.92	52.68	80.95	14.06	53.22	81.61	13.05	53.82	81.05
	Sel-inc	Log Boolean	19.09	57.09	73.11	19.34	57.79	75.47	18.24	59.35	76.59
		Log NDCG	**20.03**	**58.12**	**82.34**	**21.15**	**59.26**	**83.53**	**19.17**	**60.69**	**85.62**
		Log Recall	13.12	54.76	78.66	14.93	55.76	81.91	14.45	58.22	83.97
		PCAP	13.03	52.61	81.09	12.12	53.04	81.23	9.22	51.77	78.17
256 partitions	Batch	Log Boolean	4.95	35.44	52.65	5.22	36.35	52.71	5.34	35.50	53.05
		Log NDCG	5.38	43.8	63.86	5.55	44.15	63.4	5.73	44.27	62.98
		Log Recall	5.89	44.56	61.55	5.63	44.78	61.13	5.73	44.66	61.45
		PCAP	2.17	32.64	58.36	2.01	32.74	58.17	1.91	32.82	58.78
	Full-inc	Log Boolean	7.41	40.68	57.95	6.31	41.56	59.46	6.58	42.02	59.62
		Log NDCG	8.27	47.09	67.02	6.71	49.79	69.42	7.59	51.14	69.48
		Log Recall	8.66	48.36	65.49	6.68	49.95	67.29	7.04	51.53	68.43
		PCAP	5.13	36.13	61.36	3.83	38.71	64.75	3.04	39.27	65.07
	Sel-inc	Log Boolean	8.45	42.63	59.07	9.18	43.53	61.72	9.47	44.04	61.64
		Log NDCG	8.93	**50.12**	66.88	**9.25**	**53.18**	**75.21**	**10.11**	**57.21**	**75.26**
		Log Recall	**9.22**	49.57	**67.48**	8.21	52.86	71.26	8.94	55.19	72.91
		PCAP	4.71	36.56	61.03	2.39	37.76	63.75	2.43	37.21	62.98

queries. This cache is administered with the standard SDC strategy. A final consideration is related to network latency. Communication among processors is performed in accordance with BSP as its cost model takes the network latency into consideration.

We further describe the basics of our simulator. In a baseline strategy the work-load generated by the processing of a one-term t query, containing a posting list of global length $r \cdot k \cdot P$, can be decomposed into r iterations where each iteration involves the processing of a k-sized piece of a posting list. Assuming that the query first arrives to the broker, then this work-load can be represented by the following sequence of primitive operations:

$$\mathbf{M}(t)_P \rightarrow [\mathbf{F}(k) \,\|\, P \rightarrow \mathbf{R}(k) \,\|\, P\,]^r \rightarrow \mathbf{S}(K) \,\|\, P \rightarrow \mathbf{E}(PK).$$

In this sequence, the multicast operation (**M**) represents the part in which the broker sends the term t to P processors. Then, in parallel ($\|$), all P processors perform the fetching (**F**) from secondary memory of the k sized piece of posting list. They rank (**R**) the documents present in their piece of posting lists and the sequence **F**-**R** is repeated r times. (For more than one term, the rank operation takes into consideration the costs of intersecting the posting lists and scoring the resulting documents.) Then in parallel they send (**S**) their local top-K results to the broker. Finally the broker merges (**E**) them to produce the global top-K results for the query. These operations are properly combined. For instance, if a given posting list is already stored in the LRU posting list cache, we do not charge the cost of fetching (**F**) in that particular processor.

As a validation experiment we executed the different result caching strategies (broker-side) discussed in Gan and Suel (2009) over our log. Fig. 7(a) shows that our results precisely mimics the performance results obtained in Gan and Suel (2009). This experiment allowed us to verify that our query log was properly pre-processed and that our caching policies are correctly implemented. Fig. 7(b) shows normalized results both for real execution time and simulation time for the baseline query processing strategy. These results were obtained with the Zettair Search Engine (www.seg.rmit.edu.au/zettair) and we adjusted our simulator parameters to simulate the performance of Zettair. We installed Zettair in 64 processors of a RLX cluster and broadcast queries to all processors using TCP/IP sockets to measure query throughput. The cost of processor computation, disk accesses, communication and synchronization in our simulator was determined by performing

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

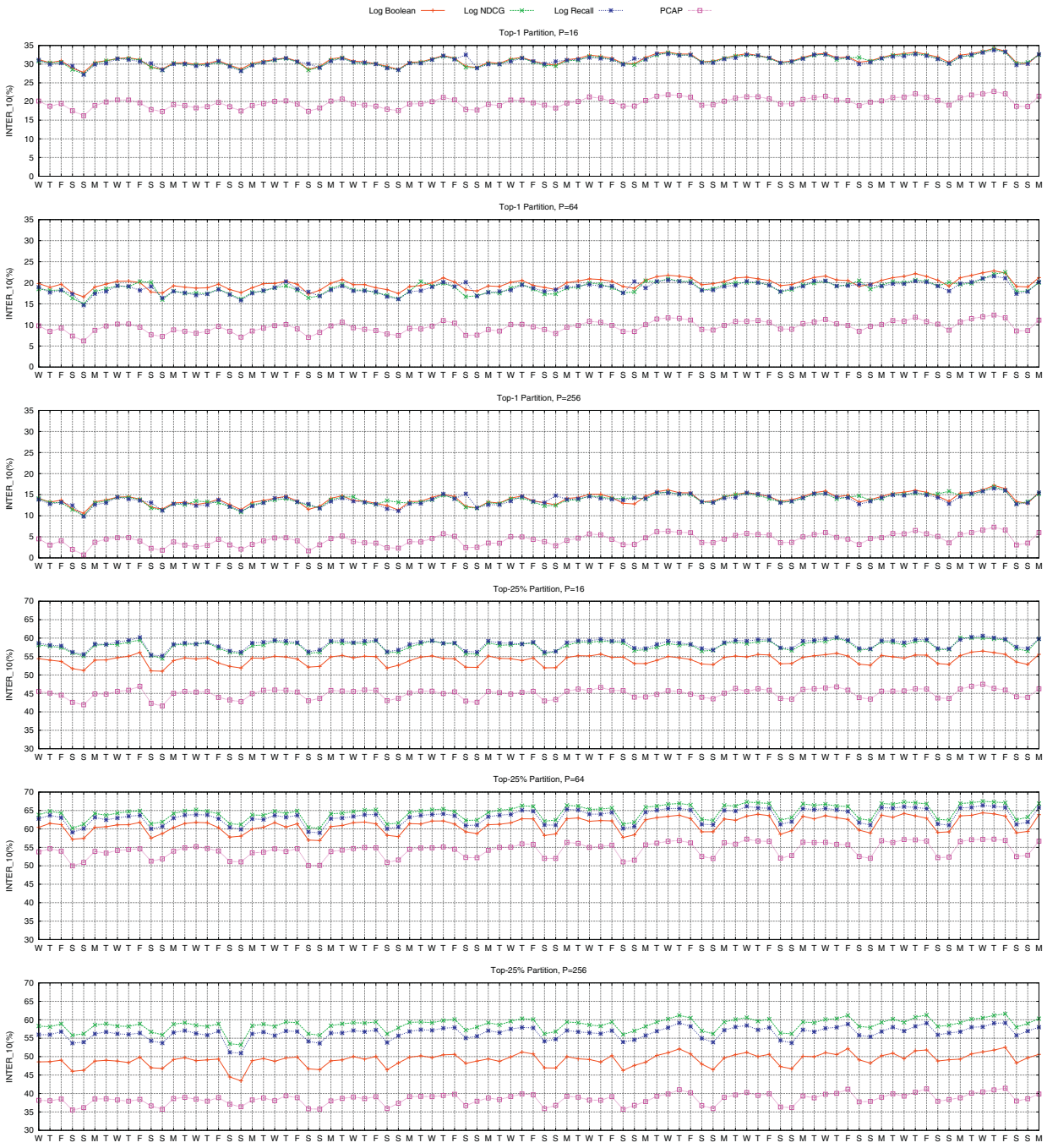

Fig. 5. Incremental learning strategies performance at $INTER_{10}$%. We used selective learning for the four routing methods.

benchmarks on RLX using the BSPonMPI library (http://bsponmpi.sourceforge.net). The results show that the simulator is capable of almost overlapping each point in the curves of Fig. 7(b).

Throughput and query answer time. In these experiments we measure throughput at regular small intervals of simulation time to detect singularities in the stream of queries. We scale up our proposal ranging P from 64 to 512 processors. Fig. 8(a) shows the query throughput obtained with different values of P. As expected, the proposed strategy is in general more efficient than the baseline strategy. However, there are some intervals in which our strategy slows throughput down to values close to those of the baseline. This happens in cases in which queries hit about the same reduced set of processors limiting the performance by the answer time of those few processors. In these cases, one should expect that the baseline strategy outperforms our proposal because it can distribute workload among the P processors, whereas our strategy is limited to a

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

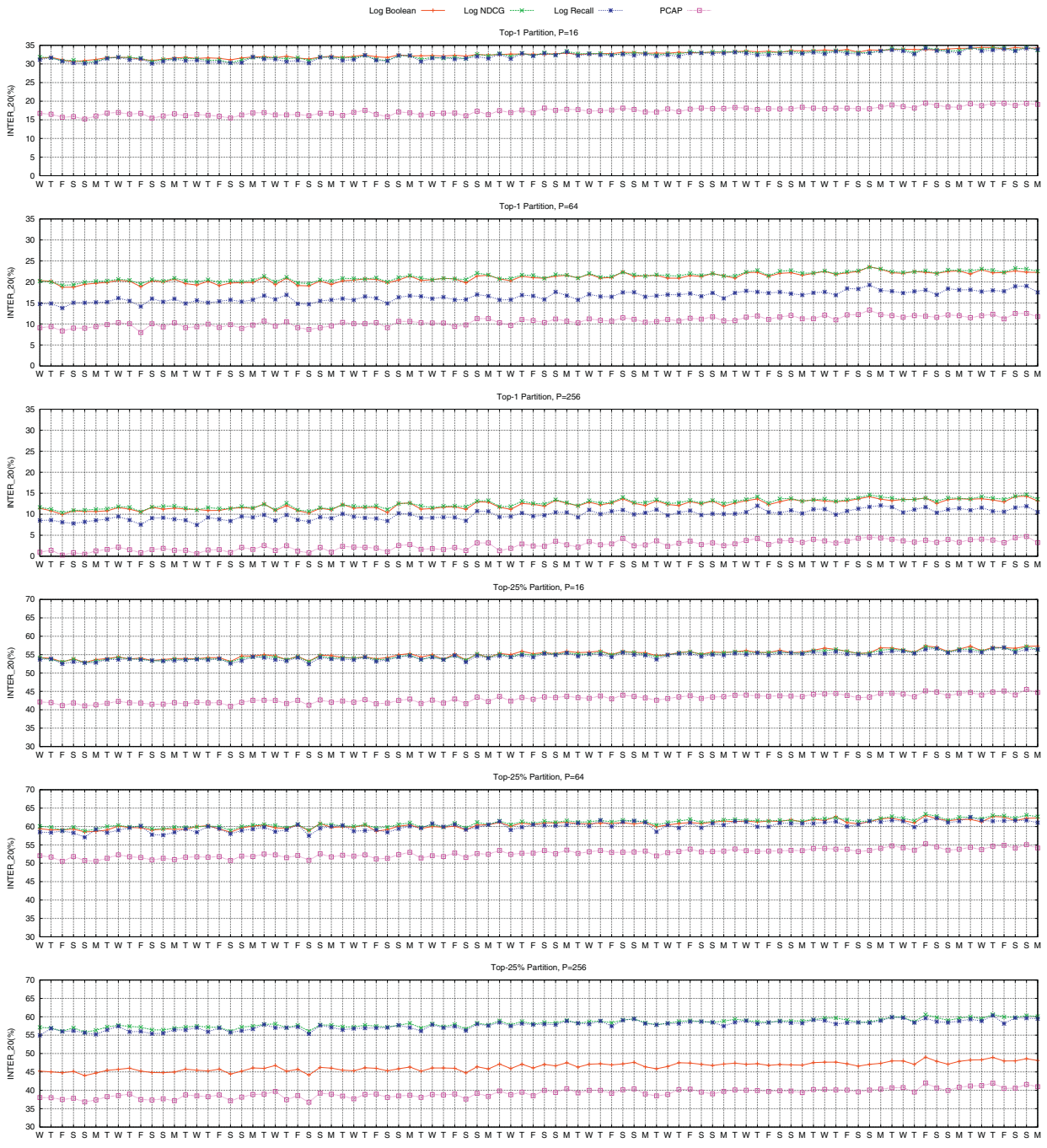

Fig. 6. Incremental learning strategies performance at $INTER_{20}\%$.

few processors. However these results show that at the same time queries routed to few processors tend to be admitted in the local posting lists caches, favoring load balancing.

Finally, we illustrate the advantages of our proposal regarding individual response query times. Fig. 8(b) presents individual response query times for $P = 128$. Similar results were achieved for the other values of P. The results show that the baseline strategy overcomes the maximum user latency by a wide margin at regular intervals because of saturation whereas the proposed strategy keeps query latency steady.

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

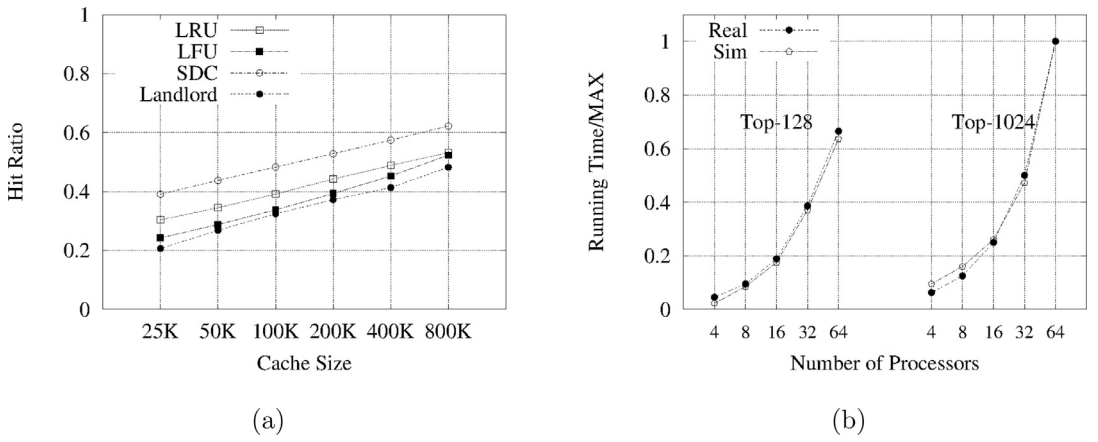

Fig. 7. Validation against third party results.

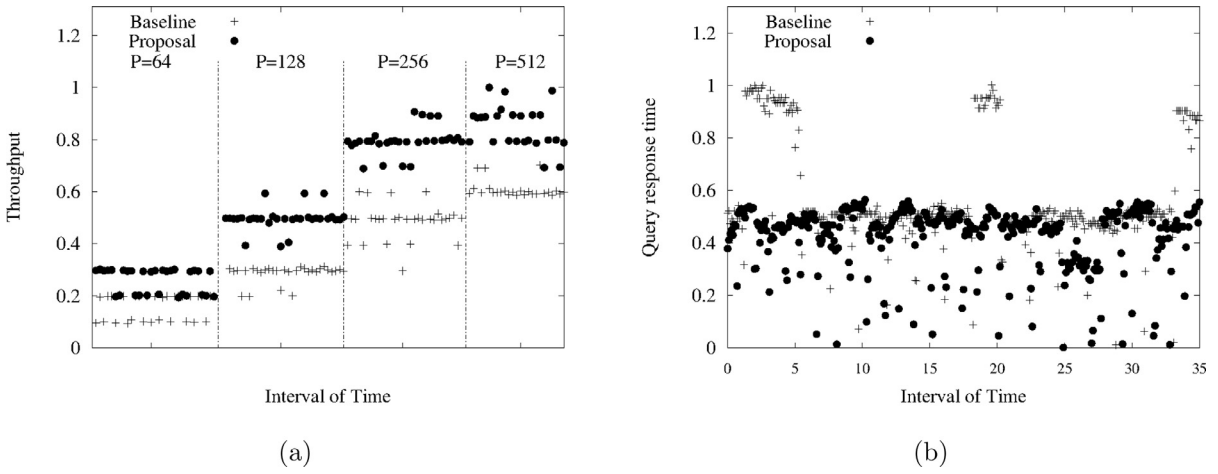

Fig. 8. Query throughput and query response time results.

7. Conclusions

We have introduced a new collection selection method for distributed search engines with vertical partitioning. The proposed method builds representations of each partition content, without using the documents content. Instead, it uses the query terms. This approach enables a learning process over a term space with lower dimensionality and greater representation, from the search engine users' point of view. The main difference between our method and PCAP is that our method models the partition lists during the learning process, using the recall gain or NDCG as a reward function for the process. This approach enables the incorporation of the objective function that needs to be optimized into the collection selection method, which significantly improves the performance.

In this study we introduce a novelty query detection method to be incorporated in the incremental learning process. The method is parameter-free and does not require tuning. It only requires definition of the significance level of the statistical test. The query selection strategy discards almost a third of all test queries, which are redundant to the model, to prevent overfitting, and improves the method generalization capabilities. The latter prevents performance deterioration compared with batch or full-learning methods.

As future work we plan to investigate how to combine query caching policies and machine learning methods to rank partitions. Caching methods are on-line as passes through cache contents on demand whereas machine learning methods are off-line as gets training data from periodically scanning logs. This resembles the concept of static and dynamic caches. As machine learning methods are able to update their internal functions in short intervals of time, they are a good alternative to quickly adapt to dynamic topic-shifts in user query contents.

As our proposal depends on logistic regression, the use of a fast implementation of the learning algorithm is advisable, reason why the use of a parallel implementation of logistic regression is a promisory line of research. There are a number of

Please cite this article as: M. Mendoza et al., Reducing hardware hit by queries in web search engines, Information Processing and Management (2016), http://dx.doi.org/10.1016/j.ipm.2016.04.008

alternatives to test the speed up of logistic regression, among them the logistic regression library provided in Spark [3] or the recently released multi-core version of liblinear (Lee, Chiang, & Lin, 2015). Regarding index update, our experiments suggest that a density-based clustering may offer advantages to document partition size balance. The use of a fast implementation of a density-based clustering algorithm appears to be another advisable research line.

Partitions need to be recalculated in the event a particular partition size shows an increment with respect to the remaining partitions. Another scenario where all partitions would need to be recalculated occurs when there is an adding of extra hardware to the search engines and therefore the number of partitions increases. In these cases, the clustering algorithm should be rerun to recalculate the partitions enforcing the replacement of a batch clustering (e.g. co-clustering) by a high dimensional incremental clustering solution. In this line, the use of hashing-based clustering strategies appears to be promisory.

Acknowledgment

Marcelo Mendoza was supported by project FONDECYT 11121435.

References

Badue, C. S., Baeza-Yates, R. A., Ribeiro-Neto, B. A., Ziviani, A., & Ziviani, N. (2007). Analyzing imbalance among homogeneous index servers in a web search system. *Information Processing and Management, 43*(3), 592–608.
Broder, A. Z., Glassman, S. C., Manasse, M. S., & Zweig, G. (1997). Syntactic clustering of the web. *Computer Networks, 29*(8-13), 1157–1166.
Callan, J., Lu, Z., & Croft, W. (1995). Searching distributed collections with inference networks. In *Sigir'95, proceedings of the 18th annual acm international conference on research and development in information retrieval. Seattle, Washington, U.S.A, July 9-13* (pp. 21–28).
Cambazoglu, B. B., Kayaaslan, E., Jonassen, S., & Aykanat, C. (2013). A term-based inverted index partitioning model for efficient distributed query processing. *ACM Transactions on the Web (TWEB), 7*(3), 15:1–15:23.
Cambazoglu, B. B., Varol, E., Kayaaslan, E., Aykanat, C., & Baeza-Yates, R. (2010). Query forwarding in geographically distributed search engines. In *Sigir'10, proceedings of the 33rd acm international conference on research and development in information retrieval, Geneva, Switzerland, July 19-23* (pp. 90–97).
Dhillon, I. S., Mallela, S., & Modha, D. S. (2003). Information-theoretic co-clustering. In *Kdd'03, proceedings of the 9th acm sigkdd international conference on knowledge discovery and data mining, Washington, DC, U.S.A, August 24-27* (pp. 19–28).
Fagni, T., Perego, R., Silvestri, F., & Orlando, S. (2006). Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Transactions on Information Systems (TOIS), 24*(1), 51–78.
Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., & Lin, C.-J. (2008). Liblinear: A library for large linear classification. *Journal of Machine Learning Research, 9*, 1871–1874.
Gan, Q., & Suel, T. (2009). Improved techniques for result caching in web search engines. In *WWW'09: Proceedings of the 18th acm international conference on world wide web, Madrid, Spain, April 20-24* (pp. 431–440).
Gravano, L., Garcia-Molina, H., & Tomasic, A. (1994). The effectiveness of gloss for the text database discovery problem. In *Sigmod'94: Proceedings of the acm sigmod international conference on management of data, Minneapolis, Minnesota, May 24-27* (pp. 126–137).
Järvelin, K., & Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS), 20*, 422–446.
Jonassen, S., & Bratsberg, S. E. (2014). Improving the performance of pipelined query processing with skipping–and its comparison to document-wise partitioning. *World Wide Web, 17*(5), 949–967.
Kruskal, W. (1958). Ordinal measures of association. *Journal of the Amercian Statistical Association, 53*(284), 814–861.
Kucukyilmaz, T., Turk, A., & Aykanat, C. (2012). A parallel framework for in-memory construction of term-partitioned inverted indexes. *The Computer Journal, 55*(11), 1317–1330.
Kulkarni, A., & Callan, J. (2010). Topic-based index partitions for efficient and effective selective search. In *Lsds-ir'10, proceedings of the 8th workshop on large-scale distributed systems for information retrieval, Geneva, Switzerland, July 23* (pp. 19–24).
Kulkarni, A., Tigelaar, A. S., Hiemstra, D., & Callan, J. (2012). Shard ranking and cutoff estimation for topically partitioned collections. In *CIKM'12, proceedings of the 21st acm international conference on information and knowledge management, Maui, Hi, U.S.A, October 29–November 2* (pp. 555–564).
Lee, M.-C., Chiang, W.-L., & Lin, C.-J. (2015). Fast matrix-vector multiplications for large-scale logistic regression on shared-memory systems. In *ICDM'15, Proceedings of the International Conference on Data Mining, Atlantic City, NJ, U.S.A, November 14–17* (pp. 835–840).
Lempel, R., & Moran, S. (2003). Predictive caching and prefetching of query results in search engines. In *Www'03, proceedings of the 12th acm international conference on World Wide Web, Budapest, Hungary, May 20–24* (pp. 19–28).
Long, X., & Suel, T. (2005). Three-level caching for efficient query processing in large web search engines. In *WWW'05, Proceedings of the 14th ACM International Conference on World Wide Web, Chiba, Japan, May 10–14* (pp. 257–266).
Lucchese, C., Orlando, S., Perego, R., & Silvestri, F. (2007). Mining query logs to optimize index partitioning in parallel web search engines. In *Infoscale '07: Proceedings of the 2nd international conference on scalable information systems* (pp. 1–9).ICST, Brussels, Belgium.
Ma, Y.-C., Chung, C.-P., & Chen, T.-F. (2011). Load and storage balanced posting file partitioning for parallel information retrieval. *Journal of Systems and Software, 84*(5), 864–884.
Moffat, A., Webber, W., & Zobel, J. (2006). Load balancing for term-distributed parallel retrieval. In *Sigir'06: Proceedings of the 29th annual acm international conference on research and development in information retrieval, Seattle, Washington, U.S.A, August 6–11* (pp. 348–355).
Puppin, D. (2007). *A search engine architecture based on collection selection*. Pisa, Italy: Department of Informatics, Pisa University Ph.D. thesis..
Puppin, D., & Silvestri, F. (2007). C++ implementation of the co-cluster algorithm by Dhillon, Mallela and Modha. http://hpc.isti.cnr.it/~diego/phd.php.
Puppin, D., Silvestri, F., & Laforenza, D. (2006). Query-driven document partitioning and collection selection. In X. Jia (Ed.), *Infoscale'06, proceedings of the 1st international conference on scalable information systems, Hong Kong, May 30–June 1* (p. 34).
Puppin, D., Silvestri, F., Perego, R., & Baeza-Yates, R. (2007). Load-balancing and caching for collection selection architectures. In *Proceedings of the 2nf international conference on scalable information systems, infoscale 2007, Suzhou, China, June 6–8, 2007: 304* (p. 2). ACM.
Puppin, D., Silvestri, F., Perego, R., & Baeza-Yates, R. (2010). Tuning the capacity of search engines: Load-driven routing and incremental caching to reduce and balance the load. *ACM Transactions on Information Systems, 28*(2).
Silvestri, F. (2007). Sorting out the document identifier assignment problem. In *Ecir'07, advances in information retrieval, 29th European conference on ir research, Rome, Italy, April 2–5* (pp. 101–112).
Valiant, L. (1990). A bridging model for parallel computation. *Communications of the ACM, 33*(8), 103–111.
Xu, J., & Croft, B. (1999). Cluster-based language models for distributed retrieval. In *Sigir'99, proceedings of the 22nd annual acm international conference on research and development in information retrieval, Berkeley, CA, U.S.A, August 15–19* (pp. 254–261).

[3] http://spark.apache.org