Mining Query Logs to Optimize Index Partitioning in Parallel Web Search Engines

Claudio Lucchese*, Salvatore Orlando*, Raffaele Perego†, Fabrizio Silvestri†

Abstract

Large-scale Parallel Web Search Engines (WSEs) needs to adopt a strategy for partitioning the inverted index among a set of parallel server nodes.

In this paper we are interested in devising an effective term-partitioning strategy, according to which the global vocabulary of terms and the associated inverted lists are split into disjoint subsets, and assigned to distinct servers. Due to the workload imbalance caused by the skewed distribution of terms in user queries, finding an effective partitioning strategy is considered a very complex task.

In this paper we first formally introduce Term Partitioning as a new optimization problem. Then we show how the knowledge mined from past WSE query logs can be used to fed the objective function of our optimization problem. In particular, the global knowledge comes from the frequent patterns extracted from past usage logs. Finally, we reports many results to show that we are able to effectively reduce both the average number of servers activated per each query, along with the workload imbalance. Experiments are conducted on large query logs of real WSEs.

Keywords: Frequent Patterns, Term Assignment, Inclusion-Exclusion Principle, Web Search Engines, Parallel Information Retrieval, Term Partitioning, Web Usage Mining, Log Analysis.

1 Introduction

Web Search Engines (WSEs) have permitted users to face Web information overload, and constitute one of the most important novelty of the last decade in the information technology field. Their design has created many new challenges in the field of computer science, one of which is to sustain the huge and exponentially growing amount of data present in the Web.

In order to improve the Web experience, it has become very important to learn about the users' behavior. This is useful not only for personalizing Web information, but also for purposes concerning the management and efficiency of Web servers.

In this paper we are interested in improving the efficiency of a large-scale parallel WSE, through the exploitation of the knowledge about its past usage, in turn obtained by mining the logs of user queries [6, 17].

WSEs use a data structure named *inverted index* for efficient retrieval of the documents satisfying a given query. Both for scalability, and high throughput, they are deployed on large clusters of servers, and therefore the inverted index has to be partitioned and each partition searched for the relevant results in parallel [3].

Due to recent proposals of a pipelined architecture, the *term partitioning* approach is now attracting some attention again [11, 13]. According to this partitioning strategy, the set of terms occurring in the index, i.e. the *lexicon*, is partitioned among the servers, and each server is able to discover only the documents containing a subset of the lexicon. The strategies proposed so far suffer from a significant load imbalance, due to the skewed distribution of terms in user queries and indexed documents.

Our first achievement is a general performance model of a term partitioned WSE. This allowed us to formally define Term Assignment as an optimization problem that aims at finding the best partitioning of the lexicon. Differently from previous works, we require a good partitioning of the lexicon not only to maximize the throughput by evenly balancing the workload, but also to minimize the average response time.

Secondly, we show that it is possible to find a trade-off between the above two goals by mining usage data from query logs. In principle, it is possible to reduce the number of queries answered by each server by assigning to the same partition those terms that often co-occur together within user queries. However, the knowledge about co-occurrence of terms is not directly required to solve the Term Assignment problem. We show that we only need to predict the number of queries that include at least one of the terms assigned to a given partition. Indeed, this turns out to be an interesting application of the Inclusion Exclusion principle [10].

Lastly, this formulation of the problem will allow

*Dip. di Informatica, Univ. Ca' Foscari di Venezia, Venezia, Italy - {clucches,orlando}@dsi.unive.it

†ISTI-CNR, Consiglio Nazionale delle Ricerche, Pisa, Italy - {r.perego,f.silvestri}@isti.cnr.it

us to propose an efficient greedy algorithm for approximating the optimum solution to the term assignment problem, by taking into consideration the most frequent terms only. Experiments, conducted on large query logs of actual WSEs, show that a good solution can be found and that the workload imbalance, which is the main drawback of a term partitioning approach, can be effectively reduced.

The rest of the paper is organized as follows. Section 2 introduces some Web information retrieval background, while Section 3 proposes a framework for evaluating term-partitioning strategies, and formally states our term-assignment problem and its objective function. Section 4 discusses in details the issues related to the knowledge that must be extracted by past query logs. In Section 5 we discuss the heuristic algorithms proposed to address the term-assignment problem. The experimental results are discussed in Section 6. Finally, Section 7 draws some conclusions and future work.

2 Web Information Retrieval Background

WSEs use an index structure called *inverted index*, which allows the efficient retrieval of documents containing the particular set of terms specified in a user query. An inverted index consists of a vocabulary where each term t is associated with an inverted list l_t storing the identities of all the documents in the collection that contain t, plus possibly some additional information about each occurrence.

In a parallel WSE, the inverted index is partitioned among a set of servers, each holding a subset of the whole index. When each sub-index is relative to a disjoint sub-collection of documents, we have a *Local*, or *Document Partitioned*, index organization. Conversely, when the whole index is split so that each partition refers to a subset of the distinct terms contained in all the documents, we have a *Global*, or *Term Partitioned*, index organization. In both cases, in front of the IR core servers, we have an additional machine hosting a *broker*, which has the task of scheduling the queries to the various servers, collecting and ranking the results, eventually producing the final list of matching documents.

Figure 2, shows the logical organization of a WSE, where k is the number of servers hosting IR core modules, q the number of terms in a given query, and r the number of ranked results returned for the query.

In a document-partitioned WSE, the broker loops taking a query out of the query stream and broadcasting it to all the p servers. Each server locally evaluates the query using its own index partition, and returns to the broker the r top-ranked documents retrieved. The broker then merges the $p \cdot r$ local answers, in order to choose the globally most relevant r results, which are finally returned to the requesting user. Since all the servers contribute to the processing of each query in parallel on their own subindexes, the load is almost evenly distributed. The processing of each query involves a communication volume proportional to $p \cdot r$, and requires $p \cdot q$ accesses to the index in order to retrieve on each server the inverted lists of the q terms of the query.

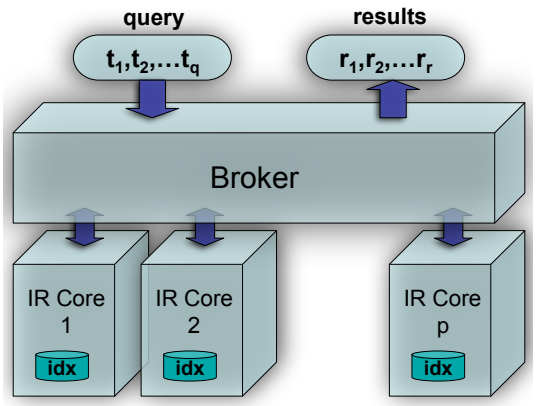

Figure 1: Organization of a parallel WSE.

In a term-partitioned WSE, for each query the broker has to select the servers holding the inverted lists relative to each of the q terms of the query. In the worst case, each one of the q inverted lists is held by a distinct server, but, on average several terms may be assigned to the same server, and the total number of servers involved is likely to be $q' \leq q$. Therefore, the broker has to split a query into q' subqueries, and send them to the corresponding servers. Each subquery is composed of the terms which are stored in the server it is sent to. When a server receives a subquery, it retrieves from the disk the inverted lists related to its subquery. In case the subquery contains more than a single term, the server intersects the associated inverted lists, and rank the partial results. Unfortunately, document ranks computed locally by each server might not be significant, since only a portion of the whole query has been considered. Thus the whole list of partial results has to be sent, in principle, to the broker. On the basis of the global knowledge of all partial results, the broker can finally perform ranking, and choose the r most relevant documents. Per–query communication volume is thus proportional to I, where I is the sum of the lengths of the inverted lists of all query terms. However, only q' servers are involved in serving the query, and the number of disk accesses is equal to the number of query terms q. It is worth noting that, with respect to the document-partitioned approach, the broker has

much more work to do, and it may become a system bottleneck.

Several papers investigated partitioning schemata for the inverted index [2–5,7–9,13,15,16]. Since in large scale IR systems disk access time may be a significant part of query-time, the term-partitioned approach is attractive because it minimizes disk activity [16]. Note that this activity takes place when the index partitions are very large with respect to the disk buffer cache available in main memory. On the other hand, the high computational load in the broker can starve the system. In addition, load imbalance may become a serious issue. Implementations of term-partitioned systems usually subdivide the vocabulary either randomly, or according to the lexicographic order of terms. In [8] the authors demonstrated that load imbalance can be mitigated by exploiting the knowledge about the frequencies of terms occurring in the queries. However, the partitioning techniques proposed so far fail in granting an even distribution of subqueries among all the servers. This is mainly because the frequencies of terms, occurring in both queries and documents, generally follow a power-law distribution. Thus, the authors noted that the servers to which many terms having high popularity are assigned may be remarkably more overloaded than those holding less of these popular terms.

Another disadvantage of term-partitioned indexes is that their management is more complex. Document-partitioned indexes can be easily and efficiently created independently by a pool of servers in a parallel environment. In the same settings, an extra processing step is instead required to exchange fragment of postings lists if we want to build a term-partitioned index [15].

In [13], Moffat *et al.* introduced a novel pipelined query evaluation methodology, based on a term-partitioned index, in which partially evaluated queries are passed through the servers that host the query terms. The experiments were conducted in a very realistic setting, making use of a large collection of data (426GB) and a long stream of actual queries. The drawback of the new method is once again the poor balancing of servers workload, which becomes a more serious problem as the number of index partitions increases. Due to the data skew present in both query term frequencies and inverted list lengths, the approach based on document-partitioning assured a better utilization of the computational resources, and outperformed their novel methodology.

Nevertheless, pipelined query evaluation seems to have a great potential. On a large set of queries generated by randomly choosing terms from the vocabulary, pipelined query evaluation outperformed by more than 20% the document-partitioned counterpart. The authors indicated as a future direction of research the exploration of load balancing methodologies based on the exploitation of term usage patterns present in the query stream. Such patterns can drive both the dynamic reassignment of lists while the query stream is being processed, and the selective replication of the most accessed inverted lists.

The load balancing problem was also addressed later in [11], where the authors exploited both term frequency information and postings list replication to improve load balancing in their pipelined WSE.

Our proposal goes exactly in the same direction, and demonstrates the feasibility and efficacy of exploiting a frequent-patterns driven partitioning of the vocabulary of terms among the servers, in order to enhance the performance of a term-partitioned, large-scale, pipelined WSE. We will show that our model overcomes the performance limits of the previous partitioning strategies.

Finally, it is worth noting that a global index organization is used also in many Peer-to-Peer environments indexing textual collections [18]. A Distributed Hash Table is usually exploited in these environments to distribute index term entries among the peers and for query routing as well. In this case a query is processed in pipeline by the involved peers in a way that resembles the proposal in [13]. Thus, also in this environment the possibility of reducing the number of peers involved by opportunely partitioning the index, has great importance.

3 Term-partitioned Parallel WSEs

Let $\mathcal{D} = \{d_1, \ldots, d_n\}$ be a set of distinct document identifiers (DocIDs), univocally associated with n documents making up a given collection, and $\mathcal{T} = \{t_1, \ldots, t_m\}$ the set of m unique terms contained in them (the lexicon). An information retrieval system will answer user queries with the help of an inverted index $\mathcal{I}$. More formally, we can define this inverted index as $\mathcal{I} = \{(t, l_t) \mid t \in \mathcal{T}, l_t \subseteq \mathcal{D}\}$, where l_t is a *postings list* that includes the DocIDs associated with all the documents that contain term t.

In this paper we are interested in term-partitioned parallel WSEs. First we introduce *p-partitioning*, a mapping function that models how the terms of the lexicon are distributed among p different servers of a term-partitioned WSE.

DEFINITION 1. *[**p-partitioning**]*
A p-partitioning *is a surjective function* $\lambda : \mathcal{T} \rightarrow \{1, \ldots, p\}$. λ *induces a partitioning* $\mathcal{T}_1, \ldots, \mathcal{T}_p$ *of* $\mathcal{T}$ *such that:*

- $\mathcal{T}_i = \{t \in \mathcal{T} \mid \lambda(t) = i\}$

- $\forall i \neq j,\ \mathcal{T}_i \cap \mathcal{T}_j = \emptyset$
- $\bigcup_{i=1,\ldots,p} \mathcal{T}_i = \mathcal{T}$

The partitioning $(\mathcal{T}_1, \ldots, \mathcal{T}_p)$, induced on $\mathcal{T}$ by λ, entails the global partitioning $(\mathcal{I}_1, \ldots, \mathcal{I}_p)$ of the inverted index $\mathcal{I}$, where $\mathcal{I}_i = \{(t,l) \in \mathcal{I} \mid t \in \mathcal{T}_i\}$. Each sub-index $\mathcal{I}_i$ is eventually assigned to one of the p different query servers of the WSE.

Table 1: Symbols used throughout the paper.

Description	Symbol
Global lexicon of terms	$\mathcal{T}$
A generic query	Q
Stream of queries submitted	Φ
Global inverted file index	$\mathcal{I}$
p-partitioning function (Def. 1)	λ
Index partition assigned to server j	$\mathcal{I}_j$
Lexicon portion assigned to server j	$\mathcal{T}_j$
Sub-query solved by server j (Def. 2)	Q_λ^j
Servers involved in serving Q (Def. 3)	H_λ^j
Width of a query Q (Def. 4)	$\omega_\lambda(Q)$
Avg. width of queries in Φ (Def. 5)	$\overline{\omega}_\lambda(\Phi)$
CPU communication overhead	$T_{overhead}$
Posting list disk transfer time	$T_{disk}(\lvert l_t \rvert)$
Computation time for each postings list	$T_{compute}(\lvert l_t \rvert)$
Workload on server j due to Q	$T_\lambda^j(Q)$
Tot. workload on server j (Def. 6)	$L_\lambda^j(\Phi)$
Tot. workload on all the servers (Def. 7)	$\overline{\overline{L}}_\lambda(\Phi)$
Avg. workload on all the servers (Def. 7)	$\overline{L}_\lambda(\Phi)$
Maximum server workload (Def. 7)	$\widehat{L}_\lambda(\Phi)$
$\lvert\{Q \in \Phi \mid \exists t \in X \subseteq \mathcal{T} \text{ s.t. } t \in Q\}\rvert$	$\rho(X)$
$\lvert\{Q \in \Phi \mid X \subseteq Q\}\rvert$, where $X \subseteq \mathcal{T}$	$\sigma(X)$

Let $Q = \{t_1, \ldots, t_q\}$, $t_i \in \mathcal{T}$, a generic query submitted to our term-partitioned WSE, where queries belong to a query stream Φ of length $|\Phi|$.

DEFINITION 2. *[**Local Sub-Query**]*
Let Q_λ^j be the subset of terms $t \in Q$ that have been mapped on $\mathcal{I}_j$. Q_λ^j can be considered as a sub-query *of Q that can locally be served by server j. More formally: $Q_\lambda^j = \{t \in Q \mid \lambda(t) = j\}$.*

DEFINITION 3. *[**Hitting Set of a Query**]*
The hitting set *$H_\lambda(Q)$ of a query Q is the set of index partitions involved in the resolution of Q in a term-partitioned WSE. More formally, $H_\lambda(Q) = \{j \mid Q_\lambda^j \neq \emptyset\}$.*

DEFINITION 4. *[**Width of a Query**]*
The width $\omega_\lambda(Q)$ of a query Q is the number of different servers involved in the resolution of Q in a term-partitioned WSE. Thus, $\omega_\lambda(Q) = |H_\lambda(Q)|$.

DEFINITION 5. *[**Average Width over a Query Stream**]*
The average width $\overline{\omega}_\lambda(\Phi)$ over a query stream Φ is the average number of different servers of a term-partitioned WSE involved in the resolution of a generic query $Q \in \Phi$ (see Def. 4). Thus, $\overline{\omega}_\lambda(\Phi) = \sum_{Q \in \Phi} \frac{\omega_\lambda(Q)}{|\Phi|}$.

3.1 Our Model of WSE Performance.

In the following discussion, we assume that our term-partitioned WSE works according to the *pipelined* approach recently proposed by Moffat, et al. [13]. Most of our considerations and assumptions are however valid also for more traditional term-partitioned WSEs.

In this novel pipeline approach, a query Q traverses all the servers of $H_\lambda(Q)$ in a pipeline fashion. Each server forwards the list of relevant DocIDs, corresponding to the portion of Q processed so far, to the next server of the pipeline. More specifically, a generic server $j \in H_\lambda(Q)$ of the pipeline receives this list, and intersects it with the posting lists associated with each term $t \in Q_\lambda^j$.

It is worth noting that, within this pipeline framework, parallelism is only exploited among different queries, and not among subsets of the same query, like in traditional term-partitioned WSEs, where all the sub-queries Q_λ^j are instead processed in parallel. Thus, since a query may need to traverse several servers, it is also important to try to reduce the average number of these servers, in order to improve single-query latency, i.e., a measure related to the query width $\omega_\lambda(Q)$. This can be done, in principle, by carefully allocating terms and associated postings lists to the p servers.

In the following we want to give an estimation of the workload incurred by a generic server and the latency of a generic query. We have to state in advance that a WSE is a complex, highly nonlinear environment, where actual query processing times are characterized by high variance and may be very hard to predict [12]. Thus, our modeling analysis can be used to understand the behavior of a WSE, and to roughly identify some average costs incurred. Our final goal is to use the estimated measures to devise an optimizing technique for term placement, aimed at improving the throughput of a WSE along with the average query latency.

Given the pair $(t, l_t) \in \mathcal{I}$, in the following we will indicate with $|l_t|$ the length of the list, and we will use the following symbols:

1. $T_{disk}(|l_t|)$, which models the time to transfer from disk the postings list l_t;
2. $T_{compute}(|l_t|)$, which corresponds to the compute time concerning the postings list l_t;
3. $T_{overhead}$, which models the CPU time spent by a

server to receive and then send a message.

Obviously the time spent for transferring data from disk, and for computing on the lists depends on the length of each list $|l_t|$.

We assume that, for each $(t, l_t) \in \mathcal{I}_j$, while t is maintained in main memory, l_t is stored on disk. According to these hypotheses, a rough estimate of the workload incurred by server $j \in H_\lambda(Q)$ to complete sub-query $Q_\lambda^j \subseteq Q$ is given by:

$$T_\lambda^j(Q) \;=\; T_{overhead} + \sum_{t \in Q_\lambda^j} \left(T_{disk}(|l_t|) + T_{compute}(|l_t|)\right).$$

In particular, $T_{disk}(|l_t|)$ may be order of magnitude larger than $T_{compute}(|l_t|)$ and $T_{overhead}$, due to some large constants like disk seek and rotation times. If this is the case, we have that the other terms become negligible:

$$T_\lambda^j(Q) \;\simeq\; \sum_{t \in Q_\lambda^j} T_{disk}(|l_t|). \tag{3.1}$$

Conversely, if we consider that, nowadays, typical sizes of main memories are huge, and that in a WSE the index $\mathcal{I}$ is partitioned to benefit from the aggregate main memory of a cluster farm, we might also assume that the OS buffer cache can fulfill most of the disk requests. In this case, we can approximate $T(Q_\lambda^j)$ as follows:

$$T_\lambda^j(Q) \;\simeq\; T_{overhead} \;+\; \sum_{t \in Q_\lambda^j} T_{compute}(|l_t|). \tag{3.2}$$

$T_{overhead}$ may be large in comparison with $T_{compute}(|l_t|)$, so it may become very significant in the above formula.

We can finally introduce some measures that can be used to estimate the throughput of a WSE when fed with a query stream Φ.

Definition 6. *[**Per Server Query-Stream Workload**]*
Given a p-partitioning λ, let $L_\lambda^j(\Phi)$ be the whole amount of work a generic server j has to deal with to process all the queries in Φ. Thus,

$$L_\lambda^j(\Phi) \;=\; \sum_{Q \in \Phi,\; Q_\lambda^j \neq \emptyset} T_\lambda^j(Q). \tag{3.3}$$

Definition 7. *[**Maximum, Total, and Average Load over a Query Stream**]*
Given the workload estimate $L_\lambda^j(\Phi)$ for each server j, we can define:

$$\widehat{L}_\lambda(\Phi) \;=\; \max_{1 \le j \le p} L_\lambda^j(\Phi) \tag{3.4}$$

$$\overline{\overline{L}}_\lambda(\Phi) \;=\; \sum_{1 \le j \le p} L_\lambda^j(\Phi) \tag{3.5}$$

$$\overline{L}_\lambda(\Phi) \;=\; \frac{\overline{\overline{L}}_\lambda(\Phi)}{p} \tag{3.6}$$

where $\widehat{L}_\lambda(\Phi)$, $\overline{\overline{L}}_\lambda(\Phi)$, and $\overline{L}_\lambda(\Phi)$ are, respectively, the maximum, total, *and* average *workload incurred by the servers of a term-partitioned WSE to process query stream Φ.*

It is worth remarking that we are supposing that, even though each query is served by a given pipeline of servers, multiple queries can be solved concurrently. Moreover, if each server has a sufficient number of distinct sub-queries Q_λ^j to answer, the communication time required to transfer queries and partial results among WSE nodes can be overlapped with useful computation. Therefore, the communication waiting times do not affect the throughput of servers, which are maintained busy. Under these assumptions, $\widehat{L}_\lambda$ corresponds to the total *completion time* to answer all the queries in Φ, and thus the average time to serve a query becomes proportional to $\widehat{L}_\lambda/|\Phi|$. Thus the following hypothesis holds.

Hypotesis 1. *In a term-partitioned WSE with a p-partitioning function λ, the* throughput *can be considered to be*

$$O\left(|\Phi|/\widehat{L}_\lambda\right).$$

Note that if we are able to balance the server workloads, by finding an appropriate $p-$partitioning function λ, this might also reduce $\widehat{L}_\lambda$, thus improving the throughput of the WSE.

On the other hand, the time elapsed to process a single query in a pipelined term-partitioned WSE is strongly influenced by communication costs. Devising a $p-$partitioning function that reduces the average width of queries (see Def. 5) results in a greater number of terms of the same query processed by the same server, and thus in a reduction of the volume of data transferred over the network. We can thus introduce the following hypothesis.

Hypotesis 2. *In a pipelined term-partitioned WSE with a p-partitioning function λ, the average time*

*(*latency*) for answering a generic query Q can be considered to be:*

$$O\left(\overline{\omega}_\lambda(\Phi)\right).$$

In a heavily loaded environment, like a large scale Web Search Engine, the throughput – i.e., the number of queries that the system is able to answer per second – is usually the most important quantity to optimize (see Hypothesis 1). From the user perspective, however, query response time is the most important figure (see Hypothesis 2). Our thesis is that it is possible to find a trade-off, and devise an optimizing technique able to optimize both. In the following Section we introduce our term assignment problem, aimed to find this trade-off.

3.2 The Term Assignment Problem With the above definitions and hypotheses we have contributed a framework regarding the performance of a pipelined term-partitioned PIRS. In this framework we are going now to introduce and formalize a novel problem relating PIRS performance and p-partitioning.

Definition 8. *[α-**weight**] Given a query stream Φ, a lexicon $\mathcal{T}$, and a p-partitioning function λ, we define α-weight on Φ according to λ as:*

$$\Omega_\lambda(\Phi) = \alpha \cdot \frac{\overline{\omega}_\lambda(\Phi)}{N_\omega} + (1-\alpha) \cdot \frac{\widehat{L}_\lambda(\Phi)}{N_L} \tag{3.7}$$

where α, $0 \leq \alpha \leq 1$, is a parameter that can be used to tune our α-weight, while N_ω and N_L are constants aimed to normalize *the two factors, so that $0 \leq \overline{\omega}_\lambda(\Phi)/N_\omega \leq 1$ and $0 \leq \widehat{L}_\lambda(\Phi)/N_L \leq 1$. Note that, in both cases 0 (1) is the best (worst) value.*

Given Hypotheses 1 and 2, we have that throughput and query response time can be improved by *minimizing* $\widehat{L}_\lambda(\Phi)$ and $\overline{\omega}_\lambda(\Phi)$, respectively.

Unfortunately, these two measures cannot be optimized independently. For example, a p-partitioning that assigns all the terms that appear in Φ to the same server, would obviously achieve the best value of $\overline{\omega}_\lambda(\Phi)$, but surely a very bad value of $\widehat{L}_\lambda(\Phi)$.

The α-weight definition allows us to consider both aspects of the problem, and devise a good tradeoff between them. Furthermore, the α parameter give us the possibility of weighting the importance of throughput and response time in optimizing the p-partitioning function.

Now we have all the figures needed to define the *Term-Assignment Problem*:

> ***The Term-Assignment Problem.***
> *Given a value α, $0 \leq \alpha \leq 1$, a query stream Φ, and p servers of a pipelined term-partitioned PIRS, the* Term-Assignment Problem *asks for finding the p-partitioning λ which minimizes $\Omega_\lambda(\Phi)$.*

4 Mining WSE query logs to solve the Term Assignment problem.

The Term Partition Problem has been expressed in terms of the query stream Φ submitted to a WSE. Unfortunately, the queries that will be actually submitted are obviously unknown when we have to partition the global index. However, several studies have confirmed the presence of self-similarities in typical WSE query logs (as an example, see [6]). We believe that the knowledge of recurrent usage patterns extracted from the queries submitted to the system in the past, can be profitably exploited also to devise an effective solution for the term-assignment problem. Hereinafter, we will continue to use the symbol Φ to simplify the notation. It will be clear from the context whether we are referring to the queries submitted in the past ($\Phi_{training}$), or to "future" queries (Φ_{test}), used to test the expected performance of the term assignments devised.

In the following we will show that both the two measures, $\overline{\omega}_\lambda(\Phi)$ and $\widehat{L}_\lambda(\Phi)$, used in Definition 8 can be expressed in terms of of $\rho(X)$, $X \subseteq \mathcal{T}$, defined as the number of queries in Φ that contain *at least one* of the terms of the term-set X, i.e., $\rho(X) = |\{Q \in \Phi \mid \exists t \in X \text{ such that } t \in Q\}|$.

This means that an interesting information to be extracted from the query log Φ are the values of $\rho(X)$, for some $X \subseteq \mathcal{T}$. We will see that there are different ways of computing $\rho(X)$ by mining Φ, and that the best technique to use depends on the heuristic strategy chosen to solve the Term Assignment problem.

4.1 Expressing the α-weight in terms of $\rho(X)$. Intuitively both $\overline{\omega}$ and $\widehat{L}$ can be optimized by taking into consideration conjunctions of terms. In fact, by assigning to the same partition terms that often co-occur together we reduce both the average width and the overhead due to extra communications.

Conversely, the following two theorems show that these two measures can be expressed in terms of ρ, thus taking into consideration only terms disjunctions and disregarding any information about terms conjunctions.

Theorem 4.1.

$$\overline{\omega}_\lambda(\Phi) = \frac{\sum_{1 \leq j \leq p} \rho(\mathcal{T}_j)}{|\Phi|}$$

Proof. Given $Q \in \Phi$, consider the following *query vector*

$\Lambda_\lambda(Q) \in \{0,1\}^p$, whose j-th component is equal to:

$$(\Lambda_\lambda)_j(Q) = \begin{cases} 1, & \text{if } Q_\lambda^j \neq \emptyset; \\ 0, & \text{otherwise.} \end{cases}$$

We can express $\omega_\lambda(Q)$ in terms of this vector, i.e.,

$$\omega_\lambda(Q) = |H_\lambda(Q)| = \sum_{1 \leq j \leq p} (\Lambda_\lambda)_j(Q)$$

Therefore, we can rewrite $\overline{\omega}_\lambda(\Phi)$ as follows:

$$\begin{aligned} \overline{\omega}_\lambda(\Phi) &= \frac{\sum_{Q \in \Phi} |H_\lambda(Q)|}{|\Phi|} \\ &= \frac{\sum_{Q \in \Phi} \sum_{1 \leq j \leq p} (\Lambda_\lambda)_j(Q)}{|\Phi|} \\ &= \frac{\sum_{1 \leq j \leq p} \left(\sum_{Q \in \Phi} (\Lambda_\lambda)_j(Q) \right)}{|\Phi|} \end{aligned}$$

Note that $\sum_{Q \in \Phi} (\Lambda_\lambda)_j(Q)$ simply corresponds to the total number of queries $Q \in \Phi$ such that $Q_\lambda^j \neq \emptyset$, i.e. the queries that contain *at least one* term belonging to $\mathcal{T}_j$, i.e., such that $\lambda(Q) = j$. Therefore:

$$\sum_{Q \in \Phi} (\Lambda_\lambda)_j(Q) = \rho(\mathcal{T}_j) \tag{4.8}$$

which proves the theorem.
□

THEOREM 4.2.

$$\begin{aligned} \widehat{L}_\lambda(\Phi) &= \max_{j=1,\ldots,p} \Big(T_{overhead} \cdot \rho(\mathcal{T}_j) + \\ &+ \sum_{t \in \mathcal{T}_j} (T_{disk}(|l_t|) + T_{compute}(|l_t|)) \cdot \rho(\{t\}) \Big) \end{aligned}$$

Proof. Recall that by definition $\widehat{L}_\lambda(\Phi)$ is the maximum workload $L_\lambda^j(\Phi)$ among the p servers defined by Equation 3.3. Consider that, for each server j, the workload $L_\lambda^j(\Phi)$ is a function of $T_{overhead}$, $T_{disk}(|l_t|)$, $T_{compute}(|l_t|)$. In particular, the server j has a global workload of:

- $T_{overhead}$ for each subquery $Q_\lambda^j \neq \emptyset$ in the query stream Φ;
- $T_{disk}(|l_t|) + T_{compute}(|l_t|)$ for each posting list l_t processed. In other words, this contribution must be considered for all the occurrences of any t, $t \in \mathcal{T}_j$, in the various subqueries Q_λ^j.

Note that the number of subqueries $Q_\lambda^j \neq \emptyset$ exactly corresponds to $\rho(\mathcal{T}_j)$. Moreover, the number of occurrences of t in all Q_λ^j is exactly $\rho(\{t\})$. This proves the theorem.
□

5 Solving the Term Assignment Problem

In order to reduce the complexity of any optimization algorithm for the Term Assignment Problem, we can consider only terms most *frequently* occurring in past Φ, counting on the self-similarity between user queries. It is worth noting that, since the distribution of terms within queries (but also in documents) follows a *power-law* distribution, not only the most frequent terms are largely the most informative ones, but also concur to realize the largest part of the WSE workload, since they likely appear in most of the queries.

Due to this feature of a typical WSE query log, in our experiments we also profitably tested the benefits of a very limited *replication* of the most frequent terms on all the parallel servers of our WSE, in combination with the optimized assignment of the remaining frequent terms.

In the following we illustrate two optimization methods. The first is based on a *local search* method. It starts from a given assignment λ, and then iteratively changes λ, thus requiring to recompute the α-weight objective function (Equation 8). We will show that to make feasible this local search method, we can profitable exploit the *Inclusion-Exclusion* (IE) principle [10], and its approximation using *Frequent Itemset Mining* (FIM) [1, 14]. The second method is instead based on a *greedy* method, which incrementally builds the λ assignment function.

The greedy method can be used to determine a first assignment λ that approximates the problem solution, while the local search can be used to further refine and optimize the initial solution.

The experiments and the results reported in the paper only refer to the greedy heuristic method, while the further refinement, made possible by the local search method, will be the subject of future work. However, we will show that the greedy method is already able to optimize the performance of a term-partitioned parallel WSE, by reducing the load imbalance and the average number of servers contacted per each query.

5.1 Local Search. Since we formulated the Term Assignment as an optimization problem, we could apply any traditional local search algorithm to handle the exponential search space of the problem, such as taboo search, simulated annealing or genetic algorithms.

After a first assignment λ of the terms, these

methods look for a new re-arrangement that improves the α-weight function as defined in Equation 8. In this case, we need to re-evaluate the various $\rho(\mathcal{T}_j)$ associated with different candidate solutions. Unfortunately, due to the size of the query log, it is not feasible to re-evaluate from scratch the objective function for each of the large number of iterations required by the above algorithms.

An interesting alternative is to calculate $\rho(\mathcal{T}_j)$ using the *Inclusion-Exclusion* (IE) principle [10]. The IE is traditionally used to compute the support of a *disjunction* of terms (items), like $\rho(\mathcal{T}_j)$, using the knowledge about the support of the *conjunctions* of terms, according to the following formula:

$$\rho(\mathcal{T}_j) \;=\; \sum_{\emptyset \neq X \subseteq \mathcal{T}_j} (-1)^{|X|+1} \sigma(X)$$

where $\sigma(X) \;=\; |\{Q \in \Phi \mid X \subset Q\}|$.

Since the number of subsets of $\mathcal{T}_j$ is $2^{|\mathcal{T}_j|}$, computing the above summation may be expensive. However, it has been shown that it is possible to have a good approximation by only considering the supports of the frequent term-sets $\mathcal{F}$ occurring at least *min_supp* times in the query log. The collection $\mathcal{F}$ can be mined by using a FIM algorithm. To this end, we consider Φ as the *input dataset* containing several *transactions/queries* Q, each made up of distinct *items/terms* in $\mathcal{T}$. By running a FIM algorithm on Φ we obtain the list of all the term-sets $X \subseteq \mathcal{T}$ that co-occur in a number of transactions/queries greater than a given threshold *min_supp*, together with their exact *support* $\sigma(X)$.

Given $\mathcal{F}$, we can finally define the IE approximation $\widetilde{\rho}(\mathcal{T}_j, \mathcal{F})$ as follows:

$$\widetilde{\rho}(\mathcal{T}_j, \mathcal{F}) \;=\; \sum_{\emptyset \neq X \subseteq \mathcal{T}_j,\; X \in \mathcal{F}} (-1)^{|X|+1} \sigma(X) \tag{5.9}$$

This approximation is likely to be enough accurate, especially with *low support* thresholds, which however makes expensive the application of this methodology, even considering the sparseness of query log.

For this reason, in this paper we only investigate in depth a greedy algorithm, which only needs the extraction of single frequent terms from Φ, i.e., the set $\mathcal{F}_1$ of all terms occurring at least *min_supp* times in Φ.

5.2 A Greedy Solution. Greedy heuristics are usually very useful to find locally optimal solutions which can be further improved by means of other optimization methods. In our specific problem, we will show that the quality of the solution found by our greedy strategy is comparably better than other solutions.

The term-assignment problem can be thought as a variation of the classical NP-hard *Bin Packing* problem, where we have to place objects of certain *weights* into a fixed number of different *bins* with the aim of optimizing a given objective function. While in the original bin packing problem we simply aim at balancing the weights assigned to the bins, in our case the *objective function* to minimize not only depends on the single weights assigned to terms (our objects), but also on the mutual assignment of terms that often co-occur in the same queries to the various index partitions. This has to be considered in order to reduce the average width of queries $\overline{\omega}_\lambda$ and the communication overheads $T_{overhead}$ on each server.

One of the issues we have to deal with is that $\Omega_\lambda(\Phi)$ is an *a-posteriori* measure. It can in fact be evaluated only after all the terms in $\mathcal{T}$ have been assigned by function λ. Due to the greedy nature of the algorithm, we instead need to evaluate Ω step by step, each time a new term is going to be assigned to a partition. In particular, we need to evaluate Ω even if only a subset $\mathcal{T}'$ of all terms, $\mathcal{T}' \subset \mathcal{T}$, has been assigned, and thus only the p-partitioning function $\lambda' : \mathcal{T}' \rightarrow \{1, \ldots, p\}$ has been defined, i.e., only a *partial solution* has been so far determined. Accordingly we introduce $\Omega_{\lambda'}(\Phi)$ to denote the *partial* objective function evaluated at each step of the algorithm.

To compute $\Omega_{\lambda'}(\Phi)$, the main issue concerns the re-computation of the various $\rho(\mathcal{T}'_j)$ when a new term t is added to $\mathcal{T}'_j$. Suppose that we maintain the set of query identifiers associated to $\mathcal{T}'_j$, i.e. $\textit{qid-set}(\mathcal{T}'_j) = \{Q \in \Phi | \mathcal{T}'_j \cap Q \neq \emptyset\}$, where $\rho(\mathcal{T}'_j) = |\textit{qid-set}(\mathcal{T}'_j)|$. It is easy to show that $\rho(\mathcal{T}'_j \cup t) = |\textit{qid-set}(\mathcal{T}'_j) \cup \textit{qid-set}(\{t\})|$.

Note that this incremental computation of $\rho(\mathcal{T}'_j)$ is cheaper than the approximation made possible by Equation 5.9, which needs the extraction of the whole collection of frequent term-sets $\mathcal{F}$ from Φ. In addition, since our algorithm only assigns the most frequent terms, we can remove any infrequent term from the query log. Due to the highly skewed distribution of queries and terms in Φ, this means to significantly reduce the size of Φ. This also reduces the space complexity of the incremental re-computation of $\rho(\mathcal{T}'_j)$, and permits to store in main memory the inverted representation of the pruned Φ, based on the various *qid-sets* associated to the frequent terms.

The pseudo-code of our greedy term-assignment algorithm is shown in Algorithm 1.

The first step of the algorithm is the discovery of the most frequent terms $\mathcal{F}_1$ in the query log Φ. In fact, our algorithm assigns the terms in $\mathcal{T}$ in two distinct phases.

The most important phase of the algorithm is the

Algorithm 1 Input: $\Phi, p, \mathcal{T}$. Output: a p-partitioning function.

$\lambda' = \emptyset$
Let $\mathcal{T}_j := \emptyset$, $1 \leq j \leq p$
Let $\mathcal{F}_1 :=$ Extract_frequent_terms (Φ, *min_supp*)
$\mathcal{T}_{\mathcal{F}} := \{t \mid t \in \mathcal{T}$ and $t \in \mathcal{F}_1\}$ {Frequent terms only}
$\overline{\mathcal{T}} := \mathcal{T} \setminus \mathcal{T}_{\mathcal{F}}$
Sort $\mathcal{T}_{\mathcal{F}}$ in descending frequency order.

{Assign the *frequent* terms}
while $\mathcal{T}_{\mathcal{F}} \neq \emptyset$ **do**
  Extract the most frequent $t \in \mathcal{T}_{\mathcal{F}}$
  $\mathcal{T}_{\mathcal{F}} := \mathcal{T}_{\mathcal{F}} \setminus \{t\}$
  for all $j,\ 1 \leq j \leq p$ **do**
    Try to assign t to $\mathcal{T}_j$, and thus try to modify λ' accordingly.
    Compute $\rho(\mathcal{T}_j \cup t)$ to evaluate $\Omega_{\lambda'}(\Phi)$
  Assign t to the server j that minimizes $\Omega_{\lambda'}(\Phi)$
  $\mathcal{T}_j := \mathcal{T}_j \cup t$, and $\lambda' := \lambda' \cup (t, j)$.

{Assign the *infrequent* terms}
while $\overline{\mathcal{T}} \neq \emptyset$ **do**
  $\overline{\mathcal{T}} := \overline{\mathcal{T}} \setminus \{t\}$
  Assign t to the most under-loaded server j ($|\mathcal{T}_j| \leq |\mathcal{T}_i|,\ 1 \leq i \leq p$)
  $\mathcal{T}_j := \mathcal{T}_j \cup t$, and $\lambda' := \lambda' \cup (t, j)$.
return λ'

Table 2: Main characteristics of the query logs used.

Query log	queries	terms	query len.	date
TodoBR	22,589,568	959,833	3.433	2001
Excite	2,477,283	419,603	3.364	Sep. 16^{th} 1997
AltaVista	7,175,648	895,792	2.507	Summer 2001

former, during which the algorithm assigns the frequent terms. Similarly to [8], we sort the frequent terms by decreasing order of frequency, and exploit this order to pick, one at the time, a term $t \in \mathcal{T}$ and assigning it to the best partition T_j chosen on the basis of objective function $\Omega_{\lambda'}(\Phi)$.

In the latter phase, the remaining terms are assigned, by trying to balance the number of terms on each server. Note that these (infrequent) terms have only little or no impact on the server workloads, since they may even not occur in the future queries.

6 Experimental Results

In order to validate our term assignment strategy, based on an heuristic greedy algorithm, we used three query logs: *TodoBR*[1], *Excite*, and *AltaVista*. These are real traces of the queries submitted to the homonymous popular search engines. Table 2 reports the main characteristics of the query logs used: the number of queries in each log, the number of different terms occurring in them, the average length of queries, and their dates. Note that *TodoBR* is a huge log of a whole year. Each record of a query log refers to a single query submitted to the web search engine for requesting a *page* of results. All query logs were preliminarily cleaned and transformed into a transactional form, where each query is simply formed by a query identifier and the list $t_1, t_2, \ldots, t_q$ of terms searched for. The terms were all converted to lower case, but we did not perform stemming nor remove stopwords. The first 2/3 of each query log ($\Phi_{training}$) were used to drive the partitioning of the index, while the last part (Φ_{test}) was used to test the partitioning obtained.

Unfortunately, we do not have real collections of documents available, coherent with the query logs, and we could not test our algorithm on a real web search engine. Therefore, we validated our approach by simulating a broker and assuming constant times for T_{disk}, $T_{compute}$, and $T_{overhead}$ disregarding the lengths of the posting lists. Our model is however sound and general, and the knowledge of the actual values of these factors could be easily taken into account during the term assignment process.

In all the tests discussed here we considered a partitioning of the index among $p = 8$ servers. We conducted tests also with different numbers of partitions. Since trends and macroscopic behaviors do not change remarkably, we will not report these results.

As baseline competitors of our *greedy* approach, we consider a *random* assignment of terms, and a simple *bin packing* strategy. In the *random* case each term was randomly assigned to one of the p servers.

The plots reported in Figure 2 show the effectiveness and flexibility of our optimization function and its α-weighting. Each plot reports, as a function of α, both the values measured on Φ_{test} of $\widehat{L}_\lambda$ (in the y-axis on the left of each plot) and of $\overline{\omega}_\lambda$ (in the y-axis on the right of each plot). Furthermore, in each plot we reported the average load $\overline{L}_\lambda$, and the baseline value of $\overline{\omega}_\lambda$ resulting from the *bin packing* heuristic. The three plots (a), (c), and (e) of the figure refer to a disk-dominant setting where the load of a server is given by Equation (3.1) with $T_{disk} = 1.0$, while the plots (b), (d), and (f) are relative to a network-dominant one where the load is given by Equation (3.2) with $T_{overhead} = 4$ and $T_{compute} = 1$.

In all the cases we can see that our approach almost evenly balances the workload among servers. In fact, the values of $\overline{L}_\lambda$ and $\widehat{L}_\lambda$ are very close for all values of α up to 0.9 (0.8 in plot 2.(a)). On the other hand, the curves $\overline{\omega}_\lambda$ decrease for increasing values of α. Note that

[1]We acknowledge Prof. Nivio Ziviani of the LATIN laboratory of the computer science department of the Federal University of Minas Gerais for providing us this query log.

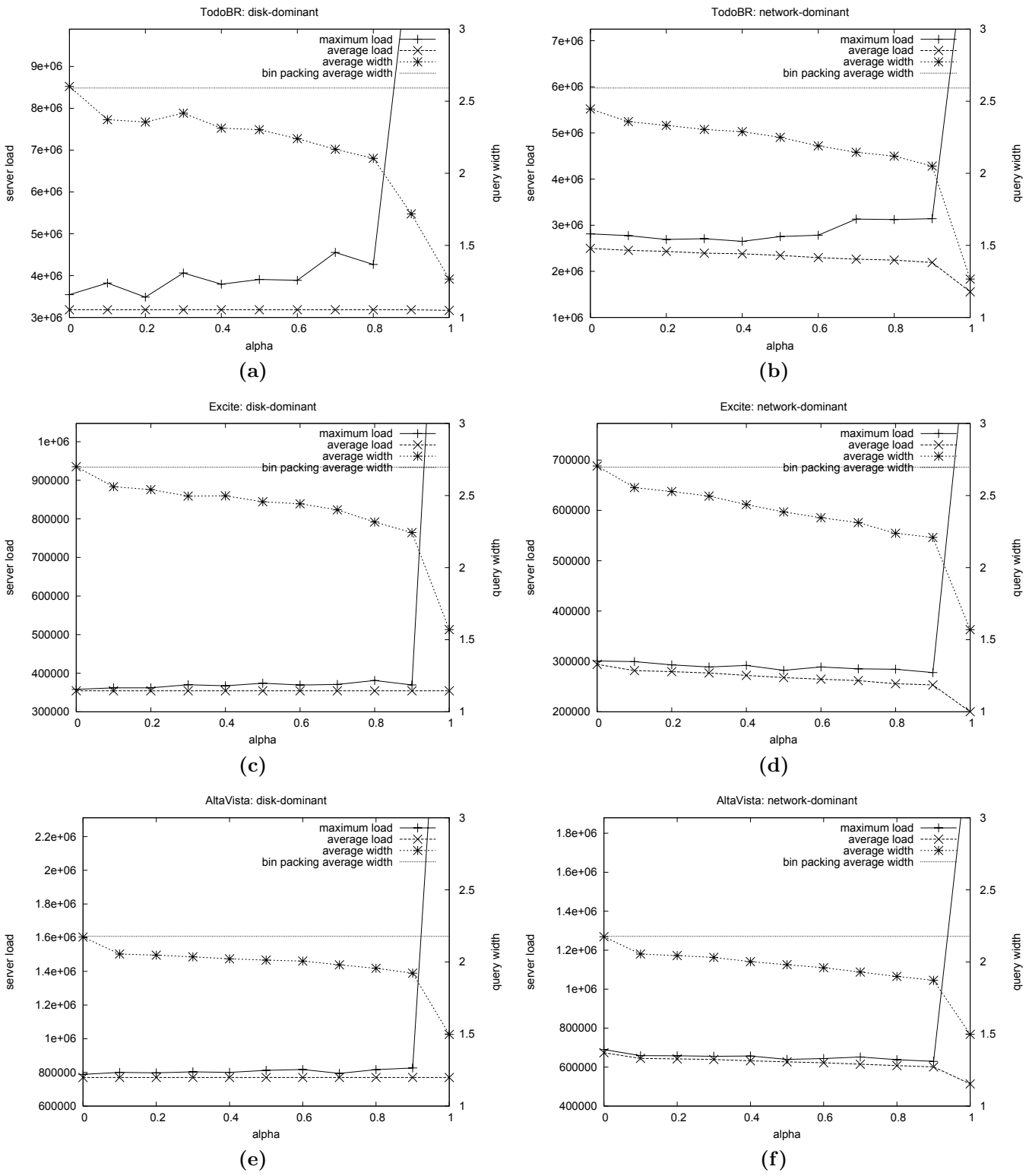

Figure 2: (a)-(f): values of $\widehat{L}_\lambda(\Phi)$ and $\overline{\omega}_\lambda(\Phi)$ as a function of the tuning parameter α;

Servers	Baseline Cases: random	Baseline Cases: bin packing	Term Assignment: $\alpha = 0.9$
$\Phi_{test} = TodoBR$			
1	28	28	50
2	31	30	20
3	17	17	14
> 3	24	25	16
$\Phi_{test} = Excite$			
1	22	22	33
2	33	33	34
3	22	21	19
> 3	23	25	14
$\Phi_{test} = AltaVista$			
1	29	29	41
2	39	39	38
3	21	21	16
> 3	11	11	5

Table 3: Percentages of queries in the three query logs as a function of the number of servers involved in their processing.

	Replication Factors					
	0.0001		0.0005		0.001	
Servers	*bin pack.*	*term ass.*	*bin pack.*	*term ass.*	*bin pack.*	*term ass.*
$\Phi_{test} = TodoBR$						
1	42	54	56	62	63	67
2	31	22	22	18	19	16
3	12	10	9	8	8	8
> 3	15	14	12	11	10	9
$\Phi_{test} = Excite$						
1	30	40	40	48	46	53
2	39	38	38	36	36	33
3	20	15	16	12	13	10
> 3	11	6	7	4	5	3
$\Phi_{test} = AltaVista$						
1	36	45	45	54	50	60
2	42	37	39	34	37	30
3	16	13	12	10	10	8
> 3	5	4	3	3	3	2

Table 4: Effect of replicating the index entries of most frequently queried terms.

on *AltaVista* the value of $\overline{\omega}_\lambda$ becomes smaller than 2, which means that the most of the queries are answered using only one server. As expected, a setting of α which weights too much $\overline{\omega}_\lambda$ results in a unbalanced assignment of terms to some server, while a setting which weights too much $\widehat{L}_\lambda$ does not improve the average query width.

From these tests we can however see that a value of α equal to 0.8 always allows our technique to devise assignments of terms resulting in a good tradeoff between WSE throughput (given Hypothesis 1) and query latency (given Hypothesis 2).

The figures reported in Table 3 show the percentage of queries occurring in the test sets Φ_{test} of the three query logs as a function of their width. Our assignment strategy allows to remarkably increase the number of queries involving only one server. Obviously, the less the servers involved in answering a query the lower the query response time and the communication volume. On the *TodoBR* log the number of queries served by a single server almost doubled w.r.t. the *random* and *bin packing* assignments. On the *Excite*, and the *AltaVista* query logs, the improvements were instead of 50% and 41%, respectively . As a consequence, the number of queries requiring more than one server decreases correspondingly. We can see that the number of queries solved by more than 3 servers is reduced by at least 1/3 on *TodoBR* and *Excite*, and it is halved in the *AltaVista* case.

Finally, we conducted some tests to measure the effect of replicating in all the servers the index entries of some of most frequently queried terms. We introduced very small percentages of replicated terms, ranging from 0.001% to 0.1% of the the total number of terms, and measured the effect on the average number of per-query servers.

The assumed behavior is the following: in the case of a multi-terms query containing some terms belonging to the set of replicated terms, these terms are all inserted in one of the sub-queries of remaining terms chosen at random. The data reported in Table 4 reports the results of these tests. We can see that replication is very effective in reducing the average number of per-query servers also in the baseline case of bin packing. However, the advantages of using our term assignment technique are remarkable also in these tests.

7 Conclusion and Future Work

We have investigated the problem of performance in term-partitioned, large-scale parallel WSE, and devised a framework to model the main quantities affecting throughput and response time in this kind of systems. Several studies demonstrated that load imbalance is one of the most important issue in term-partitioned parallel WSEs.

We thus proposed a technique for assigning the terms of the lexicon to the various index partitions in a way that allows the average number of servers activated per each query to be reduced, and the workload among the servers to be evenly balanced. We based our technique on the assumption that the knowledge of the queries submitted in the past can be profitably exploited to devise a partitioning of the index that enhance the performance of the WSE in processing future queries.

Our term partitioning algorithm is thus driven by a mining technique that extracts from query logs recurrent WSE usage patterns. These patterns succinctly enclose a global knowledge of WSE usage that is exploited in a simple greedy algorithm to assign each term of the

lexicon to the partition that minimize our cost function.

Our term assignment strategy was validated by means of simulations conducted with three real query logs of popular Web search engines. Experiments showed that the devised term assignments improved WSE throughput and query response time with respect to random and bin packing term assignments. However, since a WSE is a complex, highly nonlinear environment, we are aware that our analysis and our results have to be confirmed by testing performed on an actual parallel WSE with an actual huge index. Finally, we studied the possibility and effectiveness of replicating a small fraction of most accessed index entries on all the servers.

There are many important issues we plan to investigate in the near future: (1) experimenting with the approach on an actual test-bed system; (2) evaluating the improvements in the quality of index partitioning that can be achieved by exploiting the local search technique described in Section 3.2; (3) studying the lifetimes of our term partitioning (being driven by past usage patterns, in fact they may suffer from performance degradation due to problems concerning the freshness of data from which statistics have been drawn); (4) studying on-line policies for migrating/replicating index entries at runtime on the basis of the actual usage of the system.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. VLDB '94*, pages 487–499, September 1994.

[2] Claudine Santos Badue, Ricardo A. Baeza-Yates, Berthier A. Ribeiro-Neto, and Nivio Ziviani. Distributed query processing using partitioned inverted files. In *Proceedings of the Eighth International Symposium on String Processing and Information Retrieval (SPIRE 2001)*, pages 10–20, 2001.

[3] L. A. Barroso, J. Dean, and U. Hölze. Web search for a planet: The google cluster architecture. *IEEE Micro*, 22(2), Mar/Apr 2003.

[4] Brendon Cahoon, Kathryn S. McKinley, and Zhihong Lu. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Trans. Inf. Syst.*, 18(1):1–43, 2000.

[5] Owen de Kretser, Alistair Moffat, Tim Shimmin, and Justin Zobel. Methodologies for distributed information retrieval. In *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*, pages 66–73, Washington, DC, USA, 1998. IEEE Computer Society.

[6] T. Fagni, F. Silvestri, S. Orlando, and R. Perego. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM TOIS (Transactions on Information Systems)*, 18:21–36, 2006.

[7] Donna Harman, Wayne McCoy, Robert Toense, and Gerald Candela. Prototyping a distributed information retrieval system that uses statistical ranking. *Inf. Process. Manage.*, 27(5):449–459, 1991.

[8] Byeong-Soo Jeong and Edward Omiecinski. Inverted file partitioning schemes in multiple disk systems. *IEEE Trans. Parallel Distrib. Syst.*, 6(2):142–153, 1995.

[9] I.A. Macleod, T.P. Martin, B. Nordin, and J.R. Phillips. Strategies for building distributed information retrieval systems. *Information Processing & Management*, 6(23):511–528, 1987.

[10] Heikki Mannila and Hannu Toivonen. Multiple uses of frequent sets and condensed representations. In *Proc. of the 2nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 189–194, 1996.

[11] A. Moffat, W. Webber, and J. Zobel. Load Balancing for Term-Distributed Parallel Retrieval. In *Proceedings of the SIGIR 2006 conference*. ACM, 2006.

[12] A. Moffat and J. Zobel. What does it mean to 'measure performance'? In X. Zhou, S. Su, M. P. Papazoglou, M. E. Owlowska, and K. Jeffrey, editors, *Proceedings of the International Conference on Web Informations Systems*, pages 1–12, Brisbane, Australia, November 2004. Springer. Published as LNCS 3306.

[13] Alistair Moffat, William Webber, Justin Zobel, and Ricardo Baeza-Yates. A pipelined architecture for distributed text query evaluation. Information Retrieval, To Appear.

[14] S. Orlando, P. Palmerini, R. Perego, and F. Silvestri. Adaptive and resource-aware mining of frequent sets. In *Proc. The 2002 IEEE International Conference on Data Mining (ICDM '02)*, pages 338–345, 2002.

[15] Berthier Ribeiro-Neto, Edleno S. Moura, Marden S. Neubert, and Nivio Ziviani. Efficient distributed algorithms to build inverted files. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 105–112, New York, NY, USA, 1999. ACM Press.

[16] A. Tomasic and H. Garcia-Molina. Performance of inverted indices in shared-nothing distributed text document informatioon retrieval systems. In *PDIS '93: Proc. of the 2nd Int. Conf. on Parallel and Distributed Information Systems*, pages 8–17, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[17] Yinglian Xie and David O'Hallaron. Locality in search engine queries and its implications for caching. In *IEEE Infocom 2002*, pages 1238–1247, New York, June 2002. IEEE.

[18] Jiangong Zhang and Torsten Suel. Efficient query evaluation on large textual collections in a peer-to-peer environment. In *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, pages 225–233, 2005.