Improving Load Balance and Query Throughput of Distributed IR Systems

A. ABUSUKHON†
School of Computing and Technology
University of Sunderland, UK

M. TALIB
Department of Computer Science
University of Botswana

Abstract

As the number of queries grows over time it becomes necessary that Information Retrieval (IR) system provides high query processing rate i.e. high query throughput. In IR systems, there are three types of data partitioning, namely term-based, document-based, and hybrid partitioning. In document-based and hybrid partitioning, query is sent to all nodes and thus high level of parallelism is achieved but low query throughput. In term-based partitioning, a given query is divided into sub-queries and each sub-query is directed to the relevant node. This provides high query throughput and concurrency but poor parallelism and load balance. In this paper, the Moderate Distributed IR System (MDIRS) is proposed to improve the query throughput and load balance of hybrid partitioning. MDIRS inherits the advantage of document-based partitioning i.e. it provides moderate level of parallelism and the advantage of term-based partitioning. In other words, it provides moderate level of query throughput and load balance. Results from this paper showed that the MDIRS improved the query throughput and the total query response time of hybrid partitioning by 64% over the baseline system.

Categories and Subject Descriptors: H.1.1 [Systems and Information Theory] Information Theory - Value Information, H.3.3 [Information Search and Retrieval] Information Filtering - Retrieval Model – Search Process C.2.4 [Distributed Systems] Distributed Applications, Network Operating Systems
General Terms: Algorithm, Performance, Verification
Additional Keywords: Term Partitioning, Hybrid Partitioning, Hybrid Queries, Throughput, Load balance

IJCIR Reference Format:
A. Abusukhon, and M. Talib. Improving Load Balance and Query Throughput of Distributed IR Systems. International Journal of Computing and ICT Research, Vol. 4, No. 1, pp. 20-29. http://www.ijcir.org/volume4-number1/article3.pdf.

1. INTRODUCTION

The number of pages (documents) available online is increasing rapidly. Gulli and Signorini [2005] estimated the current size of the web. They mentioned that Google claims to index more than 8 billion pages, MSN claims about 5 billion pages and Yahoo at least 4 billion pages. They estimated the indexable

† Authors' Address: A. Abusukhon, School of Computing and Technology, University of Sunderland, UK ce4aab@student.sunderland.ac.uk; M. Talib, Department of Computer Science, University of Botswana talib@mopipi.ub.bw

International Journal of Computing and ICT Research, ISSN 1818-1139 (Print), ISSN 1996-1065 (Online), Vol.4, No.1, pp. 23 - 29, June, 2010

web to be at least 11.5 billion pages. Besides the huge document collection, we have a large number of information requests (queries) that are submitted by clients. *Sullivan*[10] reported that the number of searches per day performed by Google is 250 million.

In order to the users to effectively retrieve documents that are relevant to their needs, the IR systems must provide effective, efficient, and concurrent access to large document collections. The indices of documents must be built to perform timely information retrieval. Baeza and Ribeiro [1999] defined the index of a text as - index is a data structure built over the text to speed up the search. The most popular indices are inverted files and suffix arrays. In this research, the inverted file structure is used because of being more efficient than suffix arrays structure. Inverted files are composed of two parts - distinct tokens and the inverted lists. The inverted list consists of a number of pairs and each pair consists of document identifier in which the term appears with its frequency in that document. Other inverted file structures may contain all positions in the document at which a unique term appears, Moffat et al. [2006].

This example explains how to construct the inverted file.
Suppose that we have the following documents and terms:

Table 1: Inverted File - Example

Document identifier	Terms appear in
D1	A, B, C,A
D2	D, E, F, A
D3	C,D, B,B

In this case, the inverted file is constructed as:

A: 1:2, 2:1 B: 1:1, 3:2 C: 1:1, 3:1
D: 2:1, 3:1 E: 2:1

The inverted list 1:2, 2:1 means term A appears two times in document D1 and one time in D2. The list "1:2, 2:1" is called the inverted list.

Usually inverted files are very huge and need to be accessed at very high speed. The solution to this problem is to split the index among N nodes so that each node searches part of the distributed index.

Badue et al. [2001] evaluated distributed query performance on a real machine. They compared the local index (document partitioning) with the global index (term partitioning). They concluded that the local index provided high parallelism while the global index provided high concurrency. In local index a given query is sent to all nodes means that all nodes execute the same query. In global index partitioning, not all nodes necessarily participating in performing a single query meaning that more than one query are executed concurrently. In other words, in global index partitioning, query terms are sent only to the nodes that store their inverted list and thus allowing high concurrency.

Related Work

In general, there are three types of index partitioning namely, term-based, document-based and hybrid partitioning. In document-based partitioning, the document collection is divided equally into sub-collections then the sub-collections are distributed over nodes and then each node generates an index for that sub-collection. In this case, the documents of each sub-collection and the index reside on the same node. In term-based partitioning, all unique terms in the data collection and their full inverted lists are stored in a global index and then the terms of the global index are distributed equally among nodes using an appropriate mapping algorithm. In this paper the lexicographical mapping is used. In lexicographical mapping, each node stores a set of terms that start with a certain set of letters. For example node 1 may store all terms that start with letters A through F, and so on).

Xi et al. [2002] proposed a hybrid partitioning scheme that partitions the inverted lists into a number of chunks that are equal in size and allocated to different nodes. The main aim of hybrid partitioning is to balance the load among all the nodes. They measured the load balance and concluded that

hybrid partitioning performed better than document-based and term-based partitioning when the chunk size was small. Their results showed that for hybrid partitioning the smaller the chunk size the better the performance.

Badue et al.[2001] compared local index and global index partitioning. They concluded that the global index provided high concurrency while the local index provided high parallelism. The local index achieved better load balance than the global index and the global index performed less disks access than the local index. They used the global index to direct queries to their relevant nodes. Firstly they distributed the global index over the nodes such that all terms starting with a certain set of letters reside on one node other terms starting with different letters may reside on different nodes and so on. Secondly they directed queries to their relevant nodes by dividing query into sub-queries and send each sub-query to its relevant node.

This work differs from Xi et. al. [2002]. In their work it is difficult to direct queries to the relevant nodes because they divided each inverted list into chunks and then chunks were distributed randomly among nodes. Thus, the broker, when receives a query, it sends the query to all nodes because it has no information about the location of each term (i.e. no supper index was created). This results in low query throughput. Here, MDIRS is proposed to tackle this problem and make it easy to direct queries to their relevant nodes while the inverted lists are divided into chunks and chunks are distributed among nodes. This makes it easy to perform concurrent search as shown in Sec 3.

The main aim of this research is to investigate the effect of hybrid queries, as proposed in this paper, on query throughput, load balance, and the total query response time.

Moffat et. al. [2006] examined different methods to balance the load for term-distributed parallel architecture and proposed different techniques in order to reduce the query costs. They investigated different approaches to balance the load for a pipelined distributed retrieval system. In pipelined system, query evaluation is executed like this - suppose that we have N nodes and a query consists of three terms t1, t2, and t3 that reside on three nodes n1, n2, and n3. The query evaluation begins at n1 which retrieves the inverted list of term t1 sends it to node n2 which retrieves the inverted list of t2 and sends it with the posing list of t1 to n3 and so on. The disadvantage of this system is the load imbalance caused by a number of terms with heavily workload. They calculated the work load as $Q_t \times B_t$ where Q_t is the number of appearances of term (t) in a query batch and B_t is the inverted list length in bytes. To solve this problem they proposed to replicate those terms among nodes.

Cambazoglu et al. [2006] compared term-based and document-based partitioning with respect to query response time and query throughput. They found that document-based partitioning outperformed term-based partitioning when queries were performed sequentially. In addition, they concluded that term-based outperformed document-based partitioning when queries were performed concurrently.

The following differences are found between their work and our work:

1. We propose hybrid queries and show how they can be directed to their relevant nodes, Sec. 2.
2. We focus our work on improving the query throughput of hybrid partitioning by proposing the MDIRS. They focused their work on term-based and document-based partitioning.

System Architecture

Our system architecture consists of six nodes connected to a broker through 100 Mbps Ethernet switch. Nodes have CPU 2.80GHz and RAM 256MB.

2. RESEARCH METHODOLOGY

We perform a set of real experiments in a distributed IR system using the WT10G collection from TREC-9. First, we build the global index as follows:

1. The broker distributes the document collection among nodes document by document in round robin fashion.
2. Each node filters the document it receives from stop words and HTML symbols and then builds the inverted file in memory until a memory threshold from there the in-memory data is flushed to on-disk file [Heinz and Zobel 2003, Jaruskulchai and Kruengkrai 2002, Zobel and Moffat 2006].
3. Each node merges its on-disk files in one file.
4. Broker merges all inverted files from all nodes into one file called the global index.

In the above algorithm steps 2, 3 and 4 are performed in parallel.

After building the global index, the next step is to partition it across nodes. In term-based partitioning we distribute the global index such that all terms starting with the letters A through D reside on one node in order to allow queries to be directed to their relevant nodes.

Research Hypotheses

MDIRS relays on the following hypotheses. Hypothesis H1 states that "A h*ybrid query improves the query throughput and load balance and thus reduces the average query response time*". We define the hybrid query Hq as fusing more than one query into one query with no duplicated terms. A hybrid query is then divided into sub-queries or streams where each query stream is directed to the relevant cluster of nodes (Fig. 2 Sec. 3). Hypothesis H1 is formulated as follows –

Suppose that we have μ queries in the query buffer Qb and each query q consists of m terms t then we have:

$q_1 = t_{11}, t_{21}, \ldots, t_{m1}$
$q_2 = t_{12}, t_{22}, \ldots, t_{m2}$
.
.
$q_n = t_{1n}, t_{2n}, \ldots, t_{mn}$

where t_{mn} means term m of query n, then any hybrid query Hq may be any combination of the set T= $\{t_{11}, t_{21} \ldots, t_{mn}\}$ provided that all terms (t) in Hq are unique. Note that a hybrid query contains terms that belong to multiple queries.

Hypothesis H2 states that "*Directing query terms to the relevant cluster of nodes improve the searching time and thus reduce the average query response time of hybrid partitioning*".

In MDIRS, we divide the nodes into clusters where each cluster consists of two nodes. Each cluster of nodes stores all terms starting with a certain set of letters. For example, cluster 1 stores all terms starting with the letters A through D. The terms of the term-based index are distributed among a certain cluster of nodes such that each inverted list is divided into nearly two equal parts p1 and p2 and then p1 and p2 are distributed among the nodes of a certain cluster in round robin fashion. For example, p1 resides on node 1 of cluster 1 while p2 resides on node 2 of the same cluster.

In MDIRS hybrid queries are generated by merging a set of queries (5, 10, 25, or 50 queries) into one query then splitting this query into 3 streams where each stream contains all terms that start with a certain set of letters then streams are directed to the relevant cluster of nodes (Fig. 2 Sec 3).
So the advantages of hybrid queries are:

1. A hybrid query reduces the communication time between the broker and the nodes. Suppose we have m queries in Qb then the broker needs to send m message to all nodes. However, when hybrid query is used it is divided into n streams where n equals the number of nodes in the system and then each stream is sent to the relevant cluster of nodes. Note that n is less than m.
2. Hybrid query smoothens the skewness of the term frequency distribution of a set of queries. Jeong and Omiecinski [1995] concluded that partitioning by term resulted in load imbalance because some terms were more frequently requested in a query. Thus, nodes where these terms, associated with their inverted lists, were stored would be heavily utilized. Marin and Costa [2007] stated that load balance is sensitive to queries that include high frequency terms that refer to inverted lists of different sizes. Moffat et al. [stated that the distribution of inverted lists can be based on term frequency. They calculated the workload as L = Qt × Bt, where Qt is the number of appearance of term t and Bt is the inverted list length in bytes. They stated that load imbalance was there because of some terms with heavy workload. Our observation is that the most frequent terms of a set of queries with long inverted lists may cause the load imbalance because the nodes that store the relevant inverted lists will be heavily loaded. As a solution to this problem we propose hybrid queries. Hybrid queries reduce the Qt

value to (1) by omitting duplicated terms therefore achieving better load balance because each term t generate workload equal 1× Bt thence nodes are no longer heavily loaded.

3. THE MODERATE DISTRIBUTED IR SYSTEM [MDIRS]

In Sec 3.1 we compare the result (the total query response time) from the MDIRS with that from hybrid partitioning proposed by Xi et al. [2002]. In 3.2 we measure the total query response time when using term-partitioning scheme.

MDIRS works as below:

1. We divide the nodes into clusters such that each cluster consists of two nodes. For example, C1 consists of node 1 & node 2, C2 consists of node 3 & node 4, and C3 consists of node 5 & node 6.

2. We distribute the global index among nodes as follows: Suppose that the size of the inverted list of a given term X equals S_p and the number of nodes in each cluster C equals N_c we distribute the global index among nodes such that the inverted list of a given term X is divided into k chunks where the chunk size equals

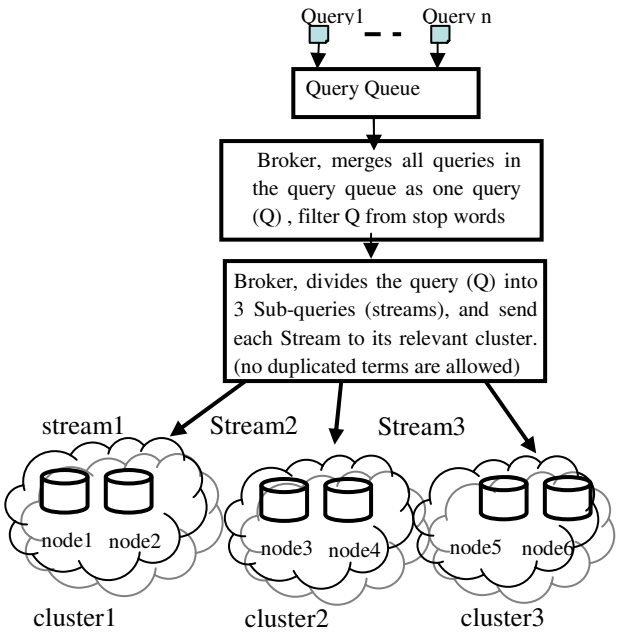
$$S_p / N_c$$

We choose the chunk size to be S_p / N_c in order to store nearly equal size of data on nodes and to make the size of each chunk as small as possible. This was because Xi et. al. [2002] concluded that hybrid partitioning performed better than document-base and term-based partitioning when chunk size was small. This was because retrieving small chunks achieved better I/O load balance. On the other hand, if they have six nodes in their system and choose the chunk size = $S_p/6$ then the inverted list of term X will be distributed among six nodes. This implies that it is difficult to direct queries to their relevant nodes because all terms with part of their inverted list may exist on all nodes and thus lead to low query throughput. To tackle this problem MDIRS assigns a set of terms that start with a certain letter(s) to a certain cluster in order to facilitate directing hybrid queries to their relevant clusters as shown in the following Table 2.

Table2: MDIRS-Term Partitioning

Cluster no.	Node no.	Letters (streams)
1	1,2	A, B, C, D, E, F, G, H, I
2	3,4	J, K, L, M, N, O, P, Q, R
3	5,6	S, T, U, V, W, X, Y, Z, Others

Fig. 2 System Architecture, MDIRS



3. We performed 50 queries extracted from TREC-9 (451-500) as a single query Q. We filtered Q from stop words and all non-digits and non-character symbols. We prohibited any duplicated terms from appearing more than once (i.e. all terms in the hybrid query are unique).
4. We divided the hybrid query Q into 3 streams because we have three clusters of nodes. Each stream consists of all terms that start with a certain set of letters. This was done in order to facilitate directing hybrid query to the relevant cluster of nodes.
5. The terms in each stream were sorted in alphabetical order and then terms were directed to one cluster. For example, stream 1 which stores all terms that start with the letters A through I is directed to cluster 1. Fig. 2 shows the system architecture and three data streams directed to the relevant clusters of nodes.

3.1 Evaluation of MDIRS

In this section we compared MDIRS and hybrid partitioning as described by Xi et. al.[2002] with respect to the total query response time. We calculated the query response time as the time elapsed between the time the broker sends a query over nodes to the time the broker receives all inverted lists from all nodes. We focused our research on improving the retrieval time, by directing queries to the relevant cluster of nodes, rather than document weighting and sorting time.

We carried out two experiments using the WT10g. In the first experiment, we distributed the global index across nodes as described by Xi et. al.[2002]. We divided the inverted list of a given term into small chunks ($k = S_p / 6$). Each chunk is sent to a certain node in round robin fashion and then we ran 50 queries sequentially. Our results showed that the total query response time was 130,233 milliseconds.

In the second experiment, we used the MDIRS. We distributed terms as shown in table 2. We ran the same 50 queries and we set μ (the number of queries used to generate the hybrid query) to 50. The total query response time was 52,469 milliseconds.

This implies that the total query response time is dropped by 0.6 or 60%. Alternatively, we can say that MDIRS improved the system throughput of Xi et. al. [2002] by 60%. This was because our algorithm split the inverted lists into chunks as Xi et. al.[2002] did but it was also able to direct hybrid queries to their relevant nodes.

3.2 Improving the Total Query Response Time of Term-based partitioning Using Hybrid Queries

In this experiment, we distributed the global index among nodes as shown in table 3 below:

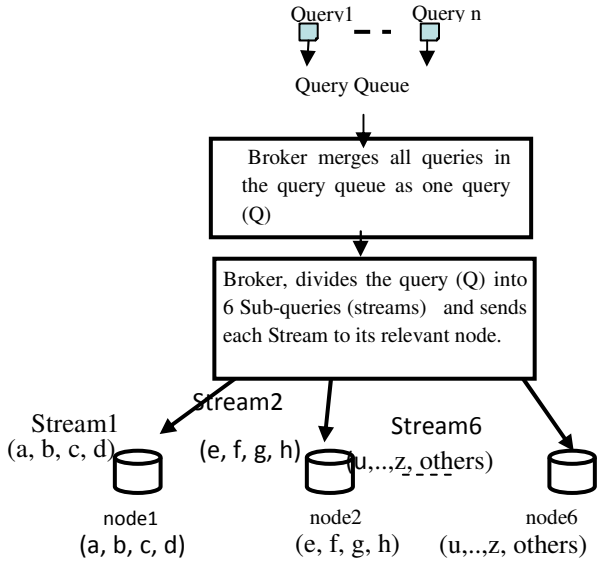
Table3: Term Partitioning

Node no.	Letters
1	A, B, C, D
2	E, F, G, H
3	I, J, K, L
4	M, N, O, P
5	Q, R, S, T
6	U, V, W, X, Y, Z, Others

As shown in Fig.3, we have six streams and each stream is directed to one node. The differences between this experiment and the experiment performed in Sec. 3.1 are given below:

1. The terms of the global index are partitioned among six nodes instead of three clusters.
2. In this experiment, nodes store full inverted lists while in the experiment performed in Sec 3.1 each node stores part of the full inverted list.

Fig. 3 System Architecture, Term Partitioning



3. In this experiment, we have six streams where each stream is directed to a different node. In other words, one node is responsible for retrieving the inverted list of a given term from disk. In the experiment performed in Sec 3.1, we create three streams, because we have three clusters of nodes, where each stream is directed to a certain cluster of nodes. In other words, two nodes participated in retrieving the full inverted list of a certain term.

We ran 50 queries and set μ to 50. The total query response time was 55,172 milliseconds. In section 3.1, we found that the total query response time of MDIRS was 52 seconds. In other words, MDIRS performs slightly better than the term-based partitioning when hybrid queries are used in both systems.

4. STUDY THE EFFECT OF INCREASING μ VALUE ON THE TOTAL QUERY RESPONSE TIME

In the previous section, we showed that hybrid query is composed of μ queries where μ = 2, 3, .. , or n queries extracted from the query buffer. In Sec 3.1 and 3.2, we set μ to 50 queries. In this section, we investigate the effect of varying μ value on the total query response time.

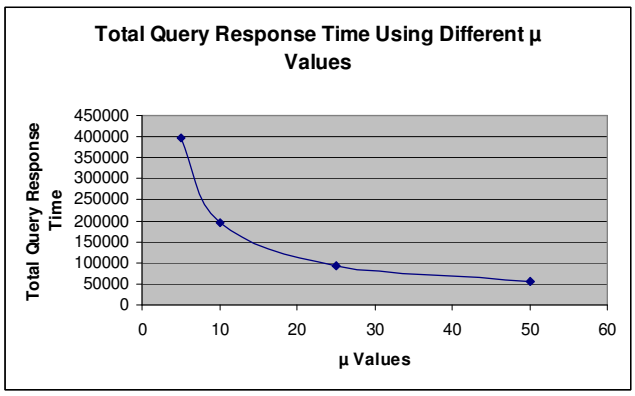
We distributed the terms of the global index as shown in Table 3 and ran a set of real experiments in order to investigate the effect of different values 5, 10, 25, and 50 of μ on the total query response time. The results are shown in Table 4 and Fig. 4.

Table4: Total Query Response Time using different μ values

μ Value (throughput factor)	**Total query response time (milliseconds)**
5	397385
10	196623
25	91579
50	55812

Table 4 and Fig.4 show that the total query response time is decreased when μ value is increased.

Fig. 4 Total Query Response Time



The above results prove research hypothesis H1.

5. STUDY THE EFFECT OF INCREASING μ VALUE ON LOAD BALANCE

In this section, we measured the load balance in terms of node utilization for different values of μ (1, 5, 10, 25, or 50 queries).

Node utilization is defined as the total amount of time the node is serving requests from the IR server divided by the total amount of time of the entire experiment, Xi et. al.[2002]. Node utilization is shown in Table 5a. The results showed that the relation between the load balance and μ is proportional relation, as described in Table 5a and Fig. 5a.

Table 5a: Node Utilization

Node label	μ =1	μ=5	μ=10	μ=25	μ=50
1	0.1606	0.0701	0.2153	0.5575	0.9137
2	0.2023	0.0969	0.2548	0.5889	0.9556
3	0.2263	0.1217	0.2937	0.6166	0.9894
4	0.2309	0.1283	0.3097	0.6271	0.9900
5	0.4326	0.1756	0.3541	0.6309	0.9936
6	0.5488	0.1839	0.3542	0.6323	0.9966

We monitored the load balance for each node in the system. We calculated the node utilization difference ΔU, where ΔU= MaxU – MinU, for each value of μ. For example when μ =1, MaxU = 0.5488 and MinU = 0.1606 thus ΔU = 0.5488 - 0.1606 or ΔU= 0.388 as described in Table 5b. Fig. 5a is drawn with the help of Table 5b.

Note that large value of ΔU means poor load balance. In general μ is increased as ΔU is decreased. In other words, we achieve better load balance (node utilization) as μ is increased.

Fig. 5a Load Balance (Node Utilization)

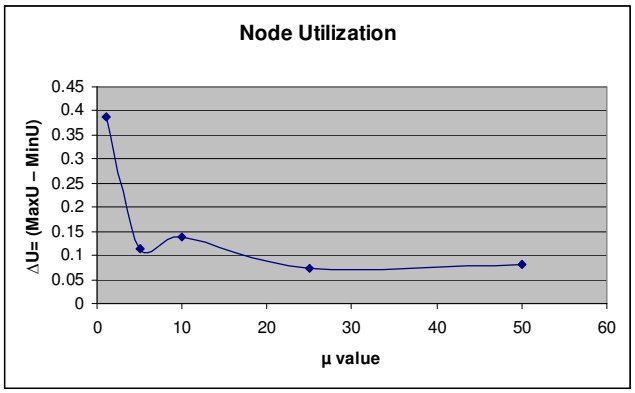


Table 5b: ΔU

μ value	ΔU= (MaxU – MinU)
1	0.388
5	0.113
10	0.138
25	0.074
50	0.082

6. CONCLUSIONS

MDIRS improved the total query response time and query throughput by 60% over hybrid partitioning. This was because MDIRS improved the searching time when queries were directed to the relevant clusters of nodes. This result proved research hypotheses H1 and H2. In addition, the results from this research showed that the relation between μ and the total query response time is reverse relation, and that the relation between μ and the node utilization is proportional relation.

7. REFERENCES

BADUE, C., BAEZA-YATES, R., RIBEIRO-NETO, B., and ZIVIANI, N., Distributed query processing using partitioned inverted files. *Proceedings.Eighth International Symposium* 2001, 10-20

BAEZA-YATES, R., and RIBEIRO-NETO, B., *Modern information retrieval* (ACM press, Addison-Wesley, New York, 1999).

CAMBAZOGLU, B.B., CATAL, A., AND AYKANAT, C., Effect of inverted index partitioning schemes on performance of query processing in parallel text retrieval systems, *ISCIS 2006, LNCS 4263*, c_Springer-Verlag Berlin Heidelberg 2006, 717–725

GULLI, A., AND SIGNORINI, A., The indexable web is more than 11.5 billion pages, *The 14th international conference on World Wide Web ACM*, New York, USA, 2005, 902-903.

HEINZ, S., AND ZOBEL, J., Efficient single-pass index construction for text databases. *Journal of the American Society for Information Science and Technology*, 2003.

JARUSKULCHAI, C., AND KRUENGKRAI, C., Building inverted files through efficient dynamic hashing. 2002.

JEONG, B.S., AND OMIECINSKI, E. (1995) Inverted File Partitioning Schemes in Multiple Disk Systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(2), pp. 142-153. IEEE Press, USA.

MARIN, M., AND COSTA, G.V. (2007) High-Performance Distributed Inverted Files. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management* CIKM'07. Lisbon, Portugal, 6-9 November 2007, pp. 935-938, ACM: New York, USA.

MOFFAT, A., WEBBER, W., AND ZOBEL, J., Load balancing for term-distributed parallel retrieval,Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, ACM, New York, USA, 2006, 348-355.

SULLIVAN, D., Searches per day. search engine *Watch*,http://searchenginewatch.com/reports/article.php/2156461, 2003.

XI, W., SOMIL, O., LUO, M., AND FOX, E., Hybrid partition inverted files for large-scale digital libraries, 2002.

ZOBEL, J., AND MOFFAT, A., Inverted files for text search engines, ACM, New York, USA, 2006.