The Journal of Systems and Software 84 (2011) 864–884

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Load and storage balanced posting file partitioning for parallel information retrieval

Yung-Cheng Ma [a,*], Chung-Ping Chung [b], Tien-Fu Chen [c]

[a] Department of Computer Science and Information Engineering, Chang-Gung University, Kwei-Shan, Tao-Yuan, Taiwan
[b] Department of Computer Science and Information Engineering, National Chiao-Tung University, Hsinchu, Taiwan
[c] Department of Computer Science and Information Engineering, National Chung-Cheng University, Chiayi, Taiwan

ARTICLE INFO

Article history:
Received 24 March 2010
Received in revised form
12 November 2010
Accepted 12 January 2011
Available online 1 February 2011

Keywords:
Load balancing
Storage balancing
Parallel information retrieval
Inverted file

ABSTRACT

Many recent major search engines on Internet use a large-scale cluster to store a large database and cope with high query arrival rate. To design a large scale parallel information retrieval system, both performance and storage cost has to be taken into integrated consideration. Moreover, a quantitative method to design the cluster in systematical way is required. This paper proposes posting file partitioning algorithm for these requirements. The partitioning follows the partition-by-document-ID principle to eliminate communication overhead. The kernel of the partitioning is a data allocation algorithm to allocate variable-sized data items for both load and storage balancing. The data allocation algorithm is proven to satisfy a load balancing constraint with asymptotical 1-optimal storage cost. A probability model is established such that query processing throughput can be calculated from keyword popularities and data allocation result. With these results, we show a quantitative method to design a cluster systematically. This research provides a systematical approach to large-scale information retrieval system design. This approach has the following features: (1) the differences to ideal load balancing and storage balancing are negligible in real-world application. (2) Both load balancing and storage balancing can be taken into integrated consideration without conflicting. (3) The data allocation algorithm is capable to deal with data items of variable-sizes and variable loads. An algorithm having all these features together is never achieved before and is the key factor for achieving load and storage balanced workstation cluster in a real-world environment.

1. Introduction

This paper studies parallel information retrieval on a cluster of workstations. The research objective is to minimize the hardware cost of the cluster to satisfy a given throughput requirement. The cluster consists of a set of identical workstations. The posting file, a data structure for information retrieval, is partitioned onto the workstations. A query is processed in parallel with the workstations. Hardware cost of the cluster depends on the cluster configuration: the number of workstations and storage capacity per workstation. Achieving the research objective lies in posting file partitioning to efficiently use the processing and storage capabilities of workstations.

Information retrieval on parallel and distributed systems has been widely studied but none of the studies has fully considered the requirements of contemporary major search engines. Previous studies (Jeong and Omiecinski, 1995; Tomasic and Molina, 1995; Riberio-Neto et al., 1998; MacFarlane et al., 2000; Moffat et al., 2006; Barroso et al., 2003; Cacheda et al., 2007; Badue et al., 2001; Lucchese et al., 2007; Moffat et al., 2007) investigated data allocation for high performance information retrieval. In these studies, storage efficiency is not considered, whereas complex simulation is required for performance evaluation. In recent years, many major search engines use a large-scale cluster to store huge amount of data and face high query arrival rate. Reducing storage cost is important and quantitative method to design a cluster is desired. This paper tackles these requirements.

This primary work is load and storage balanced posting file partitioning. The objective of the partitioning is to minimize storage requirement per workstation subject to a limited mean query processing time. Mean query processing time is estimated with popularities of keyword terms. Issues to be dealt with are

(1) load and storage balanced data allocation, and
(2) popularity-based posting file partitioning model.

The first issue is to allocate a set of items, each item being associated with a load and a data size, onto a set of workstations. The

* Corresponding author. Tel.: +886 3 2118800; fax: +886 3 2118700.
E-mail address: ycma@mail.cgu.edu.tw (Y.-C. Ma).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.01.028

objective of the data allocation is storage balancing subject to a load balancing constraint. The second issue is to reduce posting file partitioning to load and storage balanced data allocation. Storage balancing reduces the storage requirement and load balancing reduces mean query processing time. In the partitioning model, item loads are defined in terms of popularities of keyword terms. With the posting file partitioning algorithm, we show a systematic approach to determine the cluster configuration for the research objective. Contributions of this work are

(1) An asymptotically 1-optimal algorithm for load and storage balanced data allocation. This algorithm allocates variable-sized data items onto a set of workstations with solution quality on load and storage balancing been formally proved.
(2) A probabilistic posting file partitioning model to avoid communication overhead in parallel information retrieval. With this model, a query is processed in parallel without having to transfer postings between workstations. Moreover, this model formulates the posting file partitioning problem as the load and storage balanced data allocation problem. With this model, load balancing refers to maximize average throughput of the cluster of workstations.

As a result of these contributions, a quantitative method is proposed to design a clustered search engine from statistics data. Usefulness of the posting file partitioning in real-world applications is evaluated with TREC (Hardman, 1992) document collection.

This paper is organized as follows. Section 2 describes basic knowledge of information retrieval. Section 3 defines the concerned data allocation problem and describes related work on data allocation. Section 4 describes the proposed data allocation algorithm. Section 5 describes how a query is processed in parallel. Section 6 describes how the proposed data allocation algorithm is applied for posting file partitioning. Section 7 describes a quantitative method to design a cluster with the proposed posting file partitioning algorithm. Finally, conclusions are given in Section 9.

2. Background and related work

This section presents the background to devise our posting file partitioning algorithm. Section 2.1 describes the fundamental knowledge, such as the inverted file, on information retrieval. Section 2.2 presents our survey on parallel information retrieval systems.

2.1. Fundamentals on information retrieval

This section describes information retrieval concepts and analyzes its complexity to address research issues. An information retrieval system receives users queries and responds with a set of matched documents for each query. A query is a Boolean expression in which each operand is a keyword term. A document either matches or mismatches a query in a binary fashion. For each query, set operations ($\cap$, $\cup$, etc.) are performed to compute the **answer list**, which is the set of all document IDs of matched documents. The notation ANS_q denotes the answer list for query q, and the notation L_t denotes the set of all document IDs referring to documents containing term t. For the query q=(*processor* < AND> *text*), the answer list is

$$ANS_q = L_{processor} \cap L_{text},$$

which indicates the set of all documents containing both the term "*processor*" and the term "*text*".

An answer list of a query is computed using the inverted file (Frakes and Baeza-Yates, 1992; Witten et al., 1999). An example inverted file is depicted in Fig. 1. An inverted file consists of an index file and a posting file. The index file is a set of records, each containing a keyword term t and a pointer to the posting list of term t in the posting file. The posting list of term t is a sorted list of L_t. An entry in the posting list is called a posting. To process a query, the system first searches the index file and then performs set operations on the posting lists of queried terms. The set operation results can be obtained by simply merging posting lists according to Boolean operators if the posting lists are sorted (Salton, 1989).

Time complexity of query processing is as follows. Time to search the index file is no more than $O(m \times \log n)$ (Frakes and Baeza-Yates, 1992), where m is the number of queried terms and n is the number of all indexed terms. Zipf's law (Salton, 1989; Zipf, 1949) states that 95% of words in documents fall in a vocabulary with no more than 8000 distinct terms. And m is usually small. Complexity on index file side is not critical. Let f_{t_i} be the length of the posting list of a queried term t_i. The time to retrieve and merge the posting lists is $O(f_{t_1} + f_{t_2} + \cdots + f_{t_m})$. The length of a posting list increases with the size of document collection. Adding a document into the collection is to add one posting to each posting list of the terms appearing in the document. The challenge is to attack the complexity on the posting file side: We tackle this problem by proposing posting file partitioning algorithm for parallel query processing.

2.2. Related work on parallel information retrieval

This section presents our survey on parallel information retrieval. The general framework of a parallel information retrieval system with a cluster of workstations is described. The key issue to design such a parallel information retrieval system is inverted file partitioning. This section gives a brief description to previous works on inverted file partitioning.

Cacheda et al. (2007) describes a general framework of parallel information retrieval. In the general framework, a set of brokers are responsible for receiving user queries and delivering query results to users through the Internet. Upon receiving a query, a broker forwards the query to a set of query servers. The inverted file is partitioned across the query servers and the query servers work together to find out the query results. Our work is to study the inverted file partitioning scheme for parallel query processing.

The key issue in designing a parallel information retrieval system is inverted file partitioning. How the inverted file is partitioned determines how queries are processed in parallel. The system performance, both throughput and response time, depends on the inverted file partitioning. For high performance, inverted file partitioning has to balance workload and reduce communication overhead among workstations. To reduce storage cost, storage balancing is required for a homogeneous cluster. In this paper, we propose an inverted file partitioning algorithm taking integrated consideration over all these considerations.

Previous researchers (Tomasic and Molina, 1995; Jeong and Omiecinski, 1995) states that there are two ways to partition an inverted file: partition-by-term and partition-by-document scheme. With partition-by-term scheme, the partitioner finds a mapping from indexed terms to workstations. A workstation stores a subset of inverted lists from the inverted file. With partition-by-document scheme, the partitioner finds a mapping from documents to workstations. A workstation stores an inverted file covering a subset of the document collection. We briefly describe algorithms with partition-by-term scheme (Moffat et al., 2007; Lucchese et al., 2007) and algorithms with partition-by-document scheme (Cacheda et al., 2005) (Cacheda et al., 2007; Barroso et al., 2003).

Several articles (Tomasic and Molina, 1995; Jeong and Omiecinski, 1995; MacFarlane et al., 2000) (Badue et al., 2001) reported performance comparisons of the two schemes. We

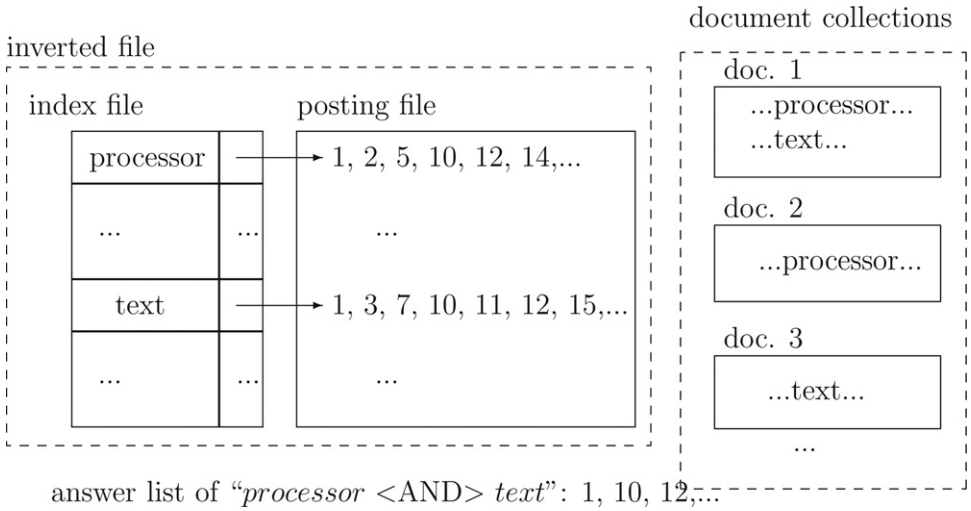


Fig. 1. Inverted file.

summarize the comparisons as follows. The advantages of partition-by-document scheme are

(1) avoid the communication overhead of transferring posting lists between workstations,
(2) easy to achieve good load balancing (MacFarlane et al., 2000), and
(3) good scalability with the increase of document collection (MacFarlane et al., 2000).

The disadvantage of partition-by-document scheme is long disk seek time to retrieve an inverted list from disks(Moffat et al., 2007).

Moffat et al. (2007) proposed load balanced partitioning algorithm with partition-by-term scheme. Workload of workstations are estimated from term popularities. The algorithm assigns inverted lists to workstations with fill-smallest policy for load balancing. Moreover, inverted lists of hot keyword terms may be replicated to multiple workstations to resolve overloading.

Lucchese et al. (2007) proposed inverted file partitioning algorithm to improve both query processing throughput and response time. The algorithm follows the partition-by-term scheme. Inverted lists are assigned to workstations according to term popularities for load balancing. To improve query response time, a clustering scheme is applied to group keyword terms frequently appearing in the same query. The algorithm then assigns inverted lists of grouped terms to the same workstation. The clustering scheme reduces the communication overhead between query servers.

Cacheda et al. (2005, 2007) proposed analytical models to analyze the performance of a cluster with partition-by-document scheme. The service rate of brokers, query servers, and network transfer are considered in the model. The effect of replicating brokers and query servers are also analyzed. However, the effect of documents-to-workstations mapping is not considered in this model. Our work builds a quantitative model to analyze the effect of document mapping scheme.

The Google search engine (Barroso et al., 2003) follows the partition-by-document scheme with replication. The inverted file is partitioned into several pieces named "index shards". An index shard covers a set of randomly chosen subset of all documents and is replicated to a pool of workstations. To process a query, the query has to be broadcast to a pool of query servers covering the whole document collection. In recent years, the indexing system is re-written with MapReduce (Dean and Ghemawat, 2008) programming scheme for better scalability.

3. Fundamentals of data allocation

Development of posting file partitioning algorithm starts from this section. This section defines the concerned data allocation problem and surveys related work. Remaining sections propose a data allocation algorithm and describe how posting file partitioning is reduced to the data allocation problem.

3.1. Load and storage balanced data allocation model

The concerned data allocation problem is as follows. The input to the data allocation algorithm is a set of data items $I = \{I_0, I_1, \ldots, I_{N-1}\}$ and a set of workstations $WS = \{WS_0, WS_1, \ldots, WS_{M-1}\}$. Each item I_i is associated with a load $Load(I_i)$ and a size s_i. We normalize the size such that

$$0 < s_i \leq 1.00 \text{ and } \max_{I_i}\{s_i\} = 1.00 \text{ for } 0 \leq i < N.$$

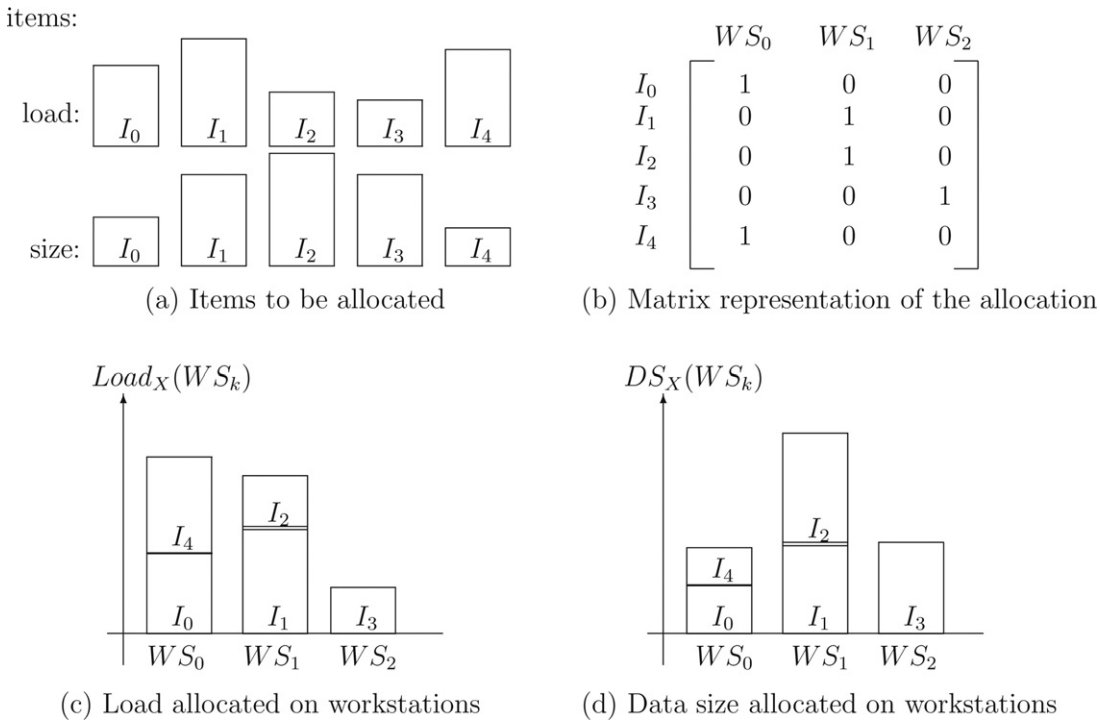
The output is an *allocation X* that allocates items in *I* onto workstations in *WS*. Replicating an item to multiple workstations is not allowed. The objective is to minimize storage requirement per workstation subject to certain load balancing requirement. We formally specify an allocation to formulate the optimization problem.

An **allocation without replication** is specified as follows. Fig. 2 depicts an example of such an allocation. An allocation is a matrix *X* in which

- each row corresponds to an item to be allocated and each column corresponds to a workstation.
- each entry is either 0 or 1, and
- there exists a unique 1 in each row of *X*.

The entry at row *i* and column *k*, denoted X_{ik}, is set to 1 if item I_i is allocated on workstation WS_k. Note that each item is allocated on a unique workstation. Load of WS_k is the total load of all items allocated on WS_k.

$$Load_X(WS_k) = \sum_{i=0}^{N-1} X_{ik} \times Load(I_i) = \sum_{I_i : X_{ik}=1} Load(I_i). \quad (1)$$



(a) Items to be allocated (b) Matrix representation of the allocation

(c) Load allocated on workstations (d) Data size allocated on workstations

Fig. 2. (a–d) Example of allocation without item replication.

Data size allocated on WS_k is the total size of all items allocated on WS_k.

$$DS_X(WS_k) = \sum_{i=0}^{N-1} X_{ik} \times s_i = \sum_{I_i : X_{ik}=1} s_i. \quad (2)$$

The objective of the allocation is storage balancing subject to a load balancing constraint. Storage balancing is to minimize the maximum amount of data allocated on a single workstation. That is, to minimize

$$\max_{WS_k}\{DS_X(WS_k)\}. \quad (3)$$

The constraint is that the load imbalancing is within the load of some item. That is,

$$Load_X(WS_k) \le \frac{L}{M} + \max_{I_i}\{Load(I_i)\} \text{ for any } WS_k, \quad (4)$$

where M is the number of workstations and L is the total load of all items.

$$L = \sum_{I_i} Load(I_i).$$

The objective is to generate an allocation X to minimize Eq. (3) subject to Eq. (4).

3.2. Related work on data allocation

Data allocation has been widely studied but none fully considered the requirements of our research objective on posting file partitioning. In early 1970s, researchers investigated data allocation for minimizing the storage cost (Johnson et al., 1974). Since 1980s, needs in high performance database systems have turned the research focus to improving the data retrieval performance (Dowdy and Foster, 1982; Wah, 1984; Wolf and Pattipati, 1990; Rotem et al., 1993; Lee and Park, 1995; Little and Venkatesh, 1995; Narendran et al., 1997; Lee et al., 2000). Starting in late 1990s, information explosion brought by the Internet raises new challenges in designing storage systems–both performance and storage cost have to be taken into consideration. Serpanos et al. (1998) proposed the MMPacking algorithm, which distributes and replicates identical-size data items onto workstations for both load and storage balancing. MMPacking (Serpanos et al., 1998) is the most closely related work but still not suitable for posting file partitioning. Requirements to achieve our research objective on posting file partitioning are

(1) optimization for both load balancing and reducing storage cost,
(2) capability in dealing with variable-size items, and
(3) no item replication.

All related work except for MMPacking fail to satisfy the first requirement. MMPacking satisfies the first requirement but not the second and the third requirements.

We propose a data allocation algorithm satisfying all these requirements. The key idea of the proposed algorithm is problem reduction to MMPacking with bin packing. We introduce MMPacking (Serpanos et al., 1998) and bin packing (Horowitz et al., 1996) in detail.

3.2.1. MMPacking

MMPacking (Serpanos et al., 1998) deals with the following optimization problem: The input is a set of n objects $B = \{B_0, B_1, \ldots, B_{n-1}\}$ with identical data sizes, and a set of M workstations $WS = \{WS_0, WS_1, \ldots, WS_{M-1}\}$. Each object B_j is associated with a load to access B_j, denoted $Load(B_j)$. The output is an *allocation with replication* that allocates objects in B onto workstations in WS. The objective is to minimize the maximum number of objects allocated on a single workstation subject to the ideal load balancing constraint.

An allocation with replication is formulated as follows. An **allocation with replication** is an $n \times M$ matrix Y similar to the allocation matrix X defined in Section 3.1, with these differences:

- each entry in Y is a real number valued between 0 and 1, and
- there may be multiple non-zero entries in a row of Y.

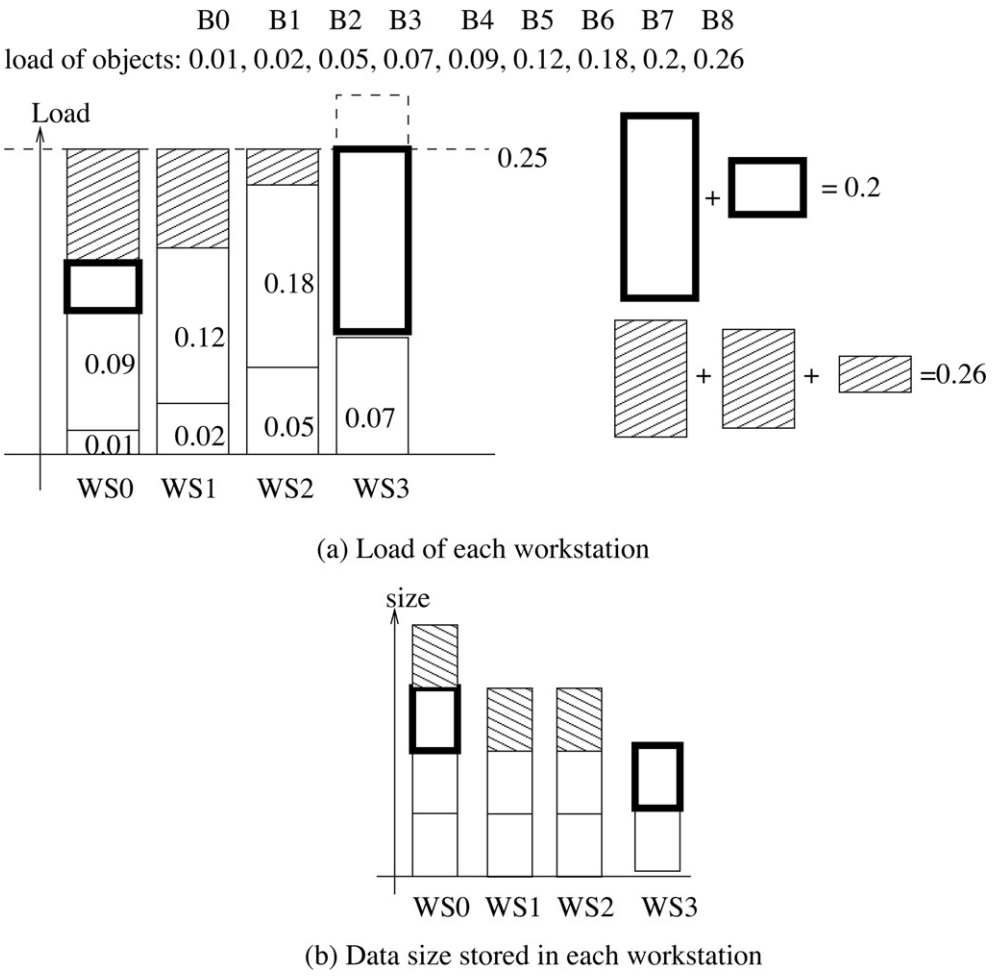


Fig. 3. (a and b) Example of MMPacking.

The entry at row j and column k, denoted Y_{jk}, represents the ratio of $Load(B_j)$ shared by workstation WS_k. The following equation holds:

$$\sum_{k=0}^{M-1} Y_{jk} = 1 \text{ for any row } j. \tag{5}$$

The load allocated on a workstation WS_k by allocation Y is as follows.

$$Load_Y(WS_k) = \sum_{j=0}^{n-1} Y_{jk} \times Load(B_j). \tag{6}$$

A row j with multiple non-zero entries means that load of accessing B_j is shared by multiple workstations. The load sharing is realized by replicating the object to multiple workstations. A copy of object B_j has to be stored in WS_k if $Y_{jk} > 0$. The number of objects allocated on WS_k by the allocation Y is $\sum_{j=0}^{n-1} \lceil Y_{jk} \rceil$.

Fig. 3 illustrates how MMPacking works. Objects are sorted in increasing order of load and then assigned to workstations in round-robin. Once the accumulated load of a workstation exceeds the ideal balanced load, part of the load of the object is split to the next workstation in round-robin. Splitting the load of an object is to replicate the object to multiple workstations. In this example, the object B_7 (with load 0.2) is replicated to WS_3 and WS_0. Replication of the object B_8 starts at WS_0, which is the last workstation to share partial load of B_7. For this example, the resultant matrix Y is shown in Fig. 4.

The following properties of MMPacking are used to analyze our proposed algorithm. Let L be the total load of all objects. Serpanos et al. (1998) have proved the following properties for MMPacking.

Property 1. *The MMPacking algorithm generates an allocation Y in which*

$$Load_Y(WS_k) = \frac{L}{M}$$

for any workstation WS_k (Serpanos et al., 1998).

Property 2. *The MMPacking algorithm allocates at least* $\lfloor n/M \rfloor$ *and at most* $\lceil n/M \rceil + 1$ *objects on a workstation* (Serpanos et al., 1998).

	WS0	WS1	WS2	WS3
B0	1.00	0.00	0.00	0.00
B1	0.00	1.00	0.00	0.00
B2	0.00	0.00	1.00	0.00
B3	0.00	0.00	0.00	1.00
B4	1.00	0.00	0.00	0.00
B5	0.00	1.00	0.00	0.00
B6	0.00	0.00	1.00	0.00
B7	0.10	0.00	0.00	0.90
B8	0.50	0.42	0.08	0.00

Fig. 4. Result matrix of MMPacking.

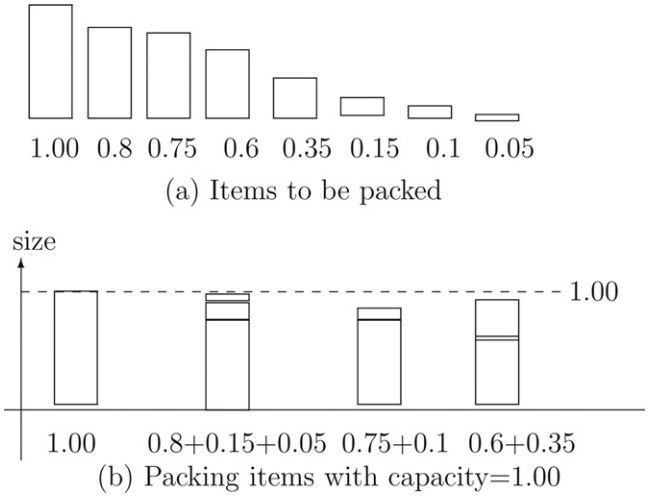


Fig. 5. (a and b) Example of bin packing.

Property 3. *In the result of MMPacking, each workstation contains at most two replicated bins. If a workstation contains two replicated bins, one of the bins is in the last workstation to share the load of the bin* (Serpanos et al., 1998).

3.2.2. Bin packing

The bin packing problem (Horowitz et al., 1996) is as follows. The input is a set of items $I = \{I_0, I_1, \ldots, I_{N-1}\}$ and a bin capacity x. Each item I_i is associated with a size s_i of it. The objective is to pack the set of items I into minimum number of bins $B = \{B_0, B_1, \ldots, B_{n-1}\}$ with capacity x. Fig. 5 depicts an example of packing items with size not exceeding 1.00 to a set of bins with capacity $x = 1.00$.

Our proposed algorithm uses the best-fit algorithm (Horowitz et al., 1996) to perform bin packing. This algorithm iteratively places an item to a bin with the smallest room left. Property 4 states the key property of the best-fit algorithm. In Section 4.2.2, we prove a guaranteed storage balancing property of our proposed algorithm based on Property 4. (See Lemma 3 for the effect of the best-fit scheme.)

Property 4. *During the best-fit bin-packing, a new bin is initialized only when the current item to be packed cannot fit into any existing bin* (Horowitz et al., 1996).

4. Load and storage balanced data allocation

This section proposes an approximate algorithm for the data allocation problem defined in Section 3.1. The proposed algorithm is outlined as follows.

Step 1: Perform bin packing to pack items in I into bins $B = \{B_0, B_1, \ldots, B_{n-1}\}$ with capacity x.

Step 2: Perform MMPacking to obtain allocation Y which allocates load of bins in B onto workstations WS. The load of a bin B_j is set as follows:

$$Load(B_j) = \sum_{I_i \in B_j} Load(I_i).$$

Step 3: **for** each B_j **do** /* allocate $I_i \in B_j$ to workstations */

- if MMPacking allocates all load of B_j to WS_k: allocate all $I_i \in B_j$ to WS_k in the final result X;
- if MMPacking replicates B_j: invoke bin splitting procedure to generate a partition $PB_j = \{B_j^{(k)} | Y_{jk} > 0\}$ where $B_j^{(k)}$ is the subset of B_j allocated on WS_k in the final result X.

The idea behind the algorithm is as follows. Step 1 packs variable-size items into bins with approximately equal size. Step 2 performs MMPacking to determine the ideal load allocation and (approximately) balance amount of bins allocated on workstations. Step 3 generates the final result X in which each item is allocated to a unique workstation. Fig. 6 depicts the final result X generated from the MMPacking result Y shown in Fig. 3. For a bin B_j not replicated by MMPacking, the whole bin is allocated to the workstation that MMPacking allocates B_j. For a bin B_j replicated by MMPacking, a bin splitting procedure is invoked to spread items in B_j to multiple workstations. Workstation WS_k is allocated a subset of B_j, denoted $B_j^{(k)}$, if MMPacking allocates partial load of B_j on WS_k ($Y_{jk} > 0$). This algorithm approximates storage balancing (when number of bins is large) since

- most of the bins are of approximately equal size, and
- each workstation contains approximately equal number of bins.

We use the notation $Load(B_j^{(k)})$ to denote total load of all items in $B_j^{(k)}$.

$$Load(B_j^{(k)}) = \sum_{I_i \in B_j^{(k)}} Load(I_i).$$

Load balancing is approximated if the bin splitting procedure in Step 3 approximates the load sharing determined by MMPacking:

$$Load(B_j^{(k)}) \approx Y_{jk} \times Load(B_j) \text{ for } WS_k : Y_{jk} > 0. \quad (7)$$

The remaining part of this section formalizes this idea. Issues to realize the idea are

(1) design of bin splitting procedure to approximate the load sharing determined by MMPacking, and
(2) selection of bin capacity x to minimize the worst-case storage requirement of a workstation.

Section 4.1 deals with the first issue and Section 4.2 deals with the second issue. Section 4.3 summarizes the discussion to form the complete algorithm.

4.1. Bin splitting for load balancing

This sub-section presents the bin splitting procedure and proves that the load balancing constraint (Eq. (4)) is satisfied. MMPacking (Serpanos et al., 1998) achieves load balancing through data replication. However, to apply for posting file partitioning, each data item has to be mapped to a unique workstation. Data replication introduces additional overhead on parallel query processing. To design a data allocation algorithm for posting file partitioning, we apply the proposed bin-splitting method to approximate load balancing without data replication.

4.1.1. Design of bin splitting procedure

The bin splitting procedure, named *SplitBin*, is shown in Fig. 7. The objective of bin splitting is to approximate the load sharing determined by MMPacking (Eq. (7)). The procedure generates a partition PB_j of bin B_j according to the MMPacking result Y. The procedure examines each item $I_i \in B_j$ and generates $B_j^{(k)}$ s in the order that MMPacking replicates B_j. A partition is made whenever

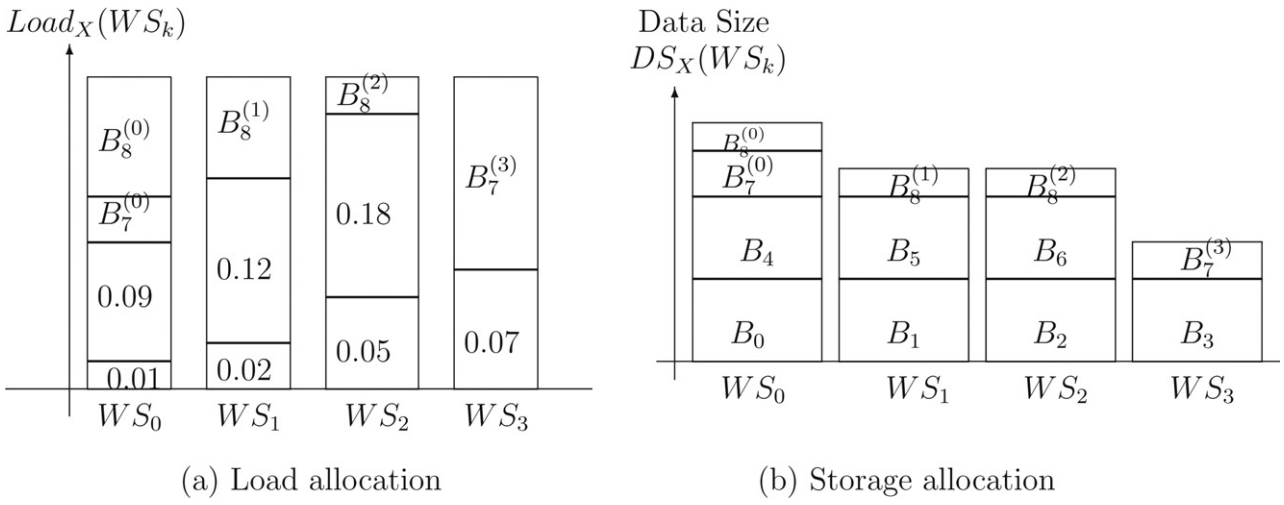


Fig. 6. (a and b) Final result generated from MMPacking result.

$Load(B_j^{(k)}) \geq Y_{jk} \times Load(B_j)$. An example is shown in Fig. 8, which depicts how the bin B_7 (with load 0.2) is split to approximate the MMPacking result shown in Fig. 3. The procedure *SplitBin* first generates $B_7^{(3)}$ and then generates $B_7^{(0)}$.

4.1.2. Analysis on load balancing property

We prove that the load balancing constraint (Eq. (4)) is satisfied after executing the bin splitting procedure. The key idea is to compare the load allocations of the final result X and the MMPacking result Y. The load of a workstation is rewritten in Corollary 1 for the comparison. Let WS_k be a workstation sharing partial load of the bin B_j in the result of MMPacking. The value $Load(B_j^{(k)})$ is compared to $Y_{jk} \times Load(B_j)$, which is the load sharing determined by MMPacking (Corollary 2 and 3). The load balancing property can then be derived (Theorem 1).

Algorithm $SplitBin(B_j, Y, PB_j)$

- Input:
  - B_j: the bin to be split
  - Y: result of MMPacking
- Output: $PB_j = \{B_j^{(k)} \subseteq B_j | Y_{jk} > 0\}$, the partition of B_j
  - $B_j^{(k)}$: the subset of B_j allocated on WS_k.
- Method:

(1) $k' \leftarrow$ the last workstation in MMPacking to share load of B_j
(2) $k \leftarrow$ the first workstation in MMPacking to share load of B_j
(3) repeat the following until $k = k'$
  (3.1) $B_j^{(k)} \leftarrow \phi$; $Load(B_j^{(k)}) \leftarrow 0$
  (3.2) **while** $Load(B_j^{(k)}) < Y_{jk} * Load(B_j)$ **and** $B_j \neq \phi$ **do**
    (3.2.1) remove an item I_i from B_j
    (3.2.2) $B_j^{(k)} \leftarrow B_j^{(k)} \cup \{I_i\}$
    (3.2.3) $Load(B_j^{(k)}) \leftarrow Load(B_j^{(k)}) + Load(I_i)$
  (3.3) $k \leftarrow (k+1) \bmod M$

Fig. 7. Bin splitting procedure.

By observing Fig. 6, the load of a workstation is rewritten as follows.

Corollary 1. *The proposed algorithm generates an allocation X in which the load of a workstation WS_k is as follows.*

$$Load_X(WS_k) = \sum_{B_j:Y_{jk}=1} Y_{jk} \times Load(B_j) + \sum_{B_j:0<Y_{jk}<1} Load(B_j^{(k)}). \quad (8)$$

Load partitioning property is as follows. Let WS_k be a workstation sharing partial load of bin B_j in the result of MMPacking. The bin splitting procedure generates items into a partition the accumulated load is greater than or equal to the load share determined by MMPacking (cf. Step (3.2) in Fig. 7). The difference between $Load(B_j^{(k)})$ and $Y_{jk} \times Load(B_j)$ is thus at most the load of the last item included in the partition (to be stated in Corollary 2). If WS_k is not the last workstation to share load of B_j, $Load(B_j^{(k)})$ exceeds (if not equal to) $Y_{jk} \times Load(B_j)$. Total load of all partitions is fixed:

$$\sum_{B_j^{(k)}} Load(B_j^{(k)}) = Load(B_j) = \sum_{WS_k} Y_{jk} \times Load(B_j).$$

Hence, as to be stated in Corollary 3, for the last workstation sharing the load of B_j, the allocated load is less than (if not equal to) the load share determined by MMPacking.

Corollary 2. *Let WS_k be a workstation sharing partial load of bin B_j in the result of MMPacking. The bin splitting procedure generates a $B_j^{(k)}$ satisfying the following equation*:

$$Load(B_j^{(k)}) \leq Y_{jk} \times Load(B_j) + \max_{I_i}\{Load(I_i)\}. \quad (9)$$

Corollary 3. *Let WS_k be the last workstation in MMPacking to share the load of bin B_j. The bin splitting procedure generates a $B_j^{(k)}$ satisfying the following equation*:

$$Load(B_j^{(k)}) \leq Y_{jk} \times Load(B_j). \quad (10)$$

Load balancing property of the proposed algorithm is as follows (where L is the total load of all items and M is the number of workstations).

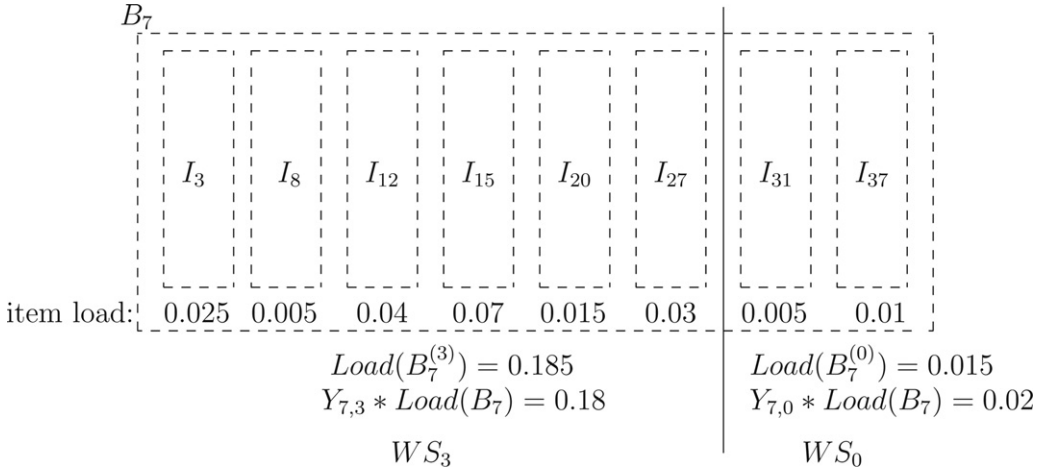


Fig. 8. Example of bin splitting.

Theorem 1 (Load balancing property). *The proposed algorithm generates an allocation X in which*

$$Load_X(WS_k) \le \frac{L}{M} + \max_{I_i}\{Load(I_i)\} \tag{11}$$

for any workstation WS_k.

Proof. The theorem is proved by rewriting Eq. (8) for various cases. In the result of MMPacking, a workstation may contain 0, 1, or 2 replicated bins (Property 3). We consider the case of a workstation contains two replicated bins.

We rewrite Eq. (8) for a workstation WS_k in which MMPacking allocates two replicated bins B_{j_1} and B_{j_2}.

$$Load_X(WS_k) = \sum_{B_j:Y_{jk}=1} Y_{jk} \times Load(B_j) + Load(B_{j_1}^{(k)}) + Load(B_{j_2}^{(k)}).$$

Property 3 states that one of the replicated bins, say B_{j_2}, is in the last workstation to share its partial load. According to Corollaries 2 and 3, we have:

$$Load(B_{j_1}^{(k)}) \le Y_{jk} \times Load(B_{j_1}) + \max_{I_i}\{Load(I_i)\},$$

$$Load(B_{j_2}^{(k)}) \le Y_{jk} \times Load(B_{j_2}).$$

Load allocated on WS_k satisfies the following equation:

$$Load_X(WS_k) \le \sum_{B_j:Y_{jk}>0} Y_{jk} \times Load(B_j) + \max_{I_i}\{Load(I_i)\}.$$

MMPacking achieves exact load balancing (Property 1).

$$\sum_{B_j:Y_{jk}>0} Y_{jk} \times Load(B_j) = \frac{L}{M}.$$

Eq. (11) is thus obtained. Proofs for other cases are similar and hence omitted. □

4.2. Bin capacity selection for storage balancing

We derive equations for selecting the bin capacity and indicate an upper bound on the allocated data size for any workstation. In selecting the bin capacity, two cases are considered: (i) setting bin capacity to the size of the largest item, (ii) setting bin capacity to be larger than the largest item.

4.2.1. Case of bin capacity being equal to largest item size

We first consider the case of setting bin capacity $x = 1$, the size of the largest item, for bin packing. Bin packing will generate a set of bins with used capacity exceeding 1/2 except for the smallest bin (see Lemma 1 below). The number of bins required will be bounded (Lemma 2), and the upper bound on allocated data size will be obtained (Theorem 2).

Lemma 1. *With bin capacity* $x = 1$, *there is at most one bin which is less than half full in the output of the bin packing.*

Proof. This lemma is proved by induction on the number of items packed. The induction hypothesis is the lemma itself. Initially, I_0 is in B_0. Suppose the lemma has not failed after packing I_i. The lemma has not failed again after the packing of I_{i+1} if no new bin is initialized for I_{i+1}. Consider the case of a new bin being initialized to pack I_{i+1}. If used capacity of each bin is at least 1/2 after the packing of I_i (see Fig. 9(a)), the new bin is the only one possible with used capacity not exceeding 1/2 after the packing of I_{i+1}. In case of there being a unique bin B_j which is less than half full after the packing of I_i (see Fig. 9(b)), according to Property 4, to use a new bin for I_{i+1} indicates that no existing bin has enough room and hence $s_{i+1} \ge 1/2$. B_j is still the only bin with used capacity not exceeding 1/2 after the packing of I_{i+1}. The lemma still has not failed after the packing of I_{i+1} for all possible cases. This argument holds until all items are packed into bins. □

With Lemma 1, upper bound on the number of bins generated can be derived. Let S be the total data size of all items,

$$S = \sum_{i=1}^{N} s_i, \tag{12}$$

where s_i is the size of item I_i. Number of bins generated is bounded as follows.

Lemma 2. *With bin capacity* $x = 1$, *the number of bins n generated by the bin packing is bounded*,

$$n \le 2 \times S + 1. \tag{13}$$

Proof. According to Lemma 1, the total size of all items is at least the total data size packed in the $(n-1)$ bins with size exceeding 1/2.

$$S \ge \frac{1}{2} \times (n-1).$$

Eq. (13) is thus obtained. □

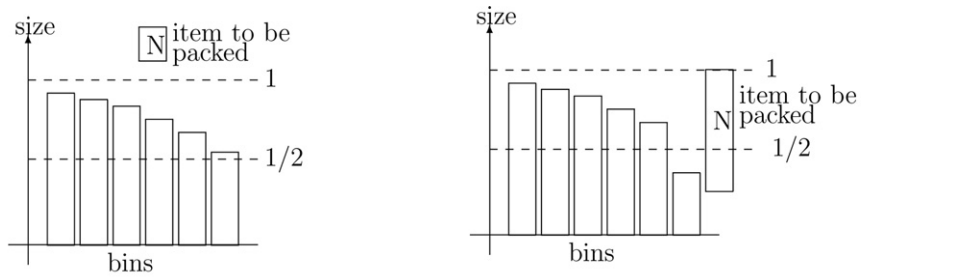


Fig. 9. (a and b) Packing an item into bins with capacity one.

Let M be the number of workstations. We derive the storage requirement of a workstation as follows.

Theorem 2 (Workstation storage requirement for bin capacity = 1). *With bin capacity $x = 1$, the proposed algorithm generates an allocation X in which*

$$DS_X(WS_k) \le 2 \times \frac{S}{M} + 3 \tag{14}$$

for any workstation WS_k.

Proof. The theorem is obtained by calculating number of bins allocated on a workstation WS_k. MMPacking allocates at most $\lceil n/M \rceil + 1$ bins in a workstation (Property 2) and the used capacity of a bin is at most 1.00. Hence we have the upper bound on the data size allocated on WS_k:

$$DS_X(WS_k) \le \left\lceil \frac{n}{M} \right\rceil + 1.$$

Total number of bins n is bounded by Eq. (13). Eq. (14) is thus obtained. □

4.2.2. Case of bin capacity being larger than item sizes

We improve storage balancing by allowing larger bin capacity. Fig. 10 shows an example of no size = 1 bins contain more than one item. (Recall that item sizes are normalized to the largest item size.) In the worst case, the most storage demanded workstation will contain twice the amount of data as those in the least storage demanded workstation. However, the worst case can be improved by enlarging bin capacity. The key issue is to select the bin capacity to minimize the (worst-case) storage requirement of a workstation.

There is a tradeoff in selecting bin capacity. Suppose the bin capacity is $x > 1.00$. Let n be the number of bins generated and M be the number of workstations. Fig. 11 shows the maximum difference in allocated data size between workstations. MMPacking (Serpanos et al., 1998) allocates from $\lfloor n/M \rfloor$ to $\lfloor n/M \rfloor + 2$ bins on each workstation. Except for the least demanded bin, the used capacity of a bin lies between $x - 1$ and x (Lemma 3). Data size allocated on a workstation is bounded as follows from the above:

$$\max_{WS_k}\{DS_X(WS_k)\} \le \left(\left\lfloor \frac{n}{M} \right\rfloor + 2 \right) \times x.$$

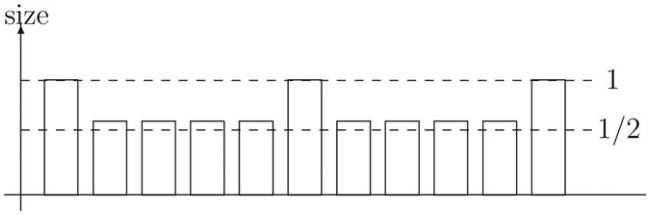


Fig. 10. Worst case of setting bin capacity to be equal to the largest item.

In a workstation, there are at most three bins with used capacity not exceeding $x - 1$: one bin resulted from bin packing and two from bin splitting. Data size allocated on a workstation is bounded as follows from the below:

$$\min_{WS_k}\{DS_X(WS_k)\} \ge \left(\left\lfloor \frac{n}{M} \right\rfloor - 3 \right) \times (x - 1).$$

The maximum difference in allocated data size between workstations is as follows:

$$\max_{WS_k}\{DS_X(WS_k)\} - \min_{WS_k}\{DS_X(WS_k)\} = O\left(\frac{n}{M}\right) + O(x). \tag{15}$$

Selecting a large x reduces number of bins n generated and hence $O(n/M)$ in Eq. (15). However, selecting a large x increases $O(x)$ in Eq. (15). The tradeoff is resolved analytically.

How to select x is outlined as follows. Lemma 3 below states packed bin sizes. Lemma 4 bounds number of generated bins according to packed bin sizes. With the bound on number of generated bins, Lemma 5 relates storage requirement to selected bin capacity, and defines the *storage requirement function* (Eq. (18)). The bin capacity x is selected to minimize the storage requirement function. The storage requirement for a workstation can then be derived (Theorem 3).

Lemma 3. *With bin capacity $x > 1$, there exists at most one bin with used capacity less than $x - 1$ in the result of bin packing.*

Proof. This lemma is proved by induction on the number of items packed. The induction hypothesis is the lemma itself. The initial condition, condition after packing of item I_0, is trivial. Suppose the lemma holds after the packing of I_i. Two possibilities exist here: (i) used capacity of each bin exceeds $x - 1$ (Fig. 12(a)), and (ii) there is only one bin B_j with used capacity not exceeding $x - 1$ (Fig. 12(b)). For case (i), after packing of I_{i+1}, the new bin (if initialized) is the only possible one with used capacity less than $x - 1$. For case (ii), no new bin will be initialized (Property 4) since size $s_{i+1} \le 1$ and B_j has enough room for I_{i+1}, and afterwards B_j is the only possible bin with used capacity not exceeding $x - 1$. The lemma thus holds again after the packing of I_{i+1}. □

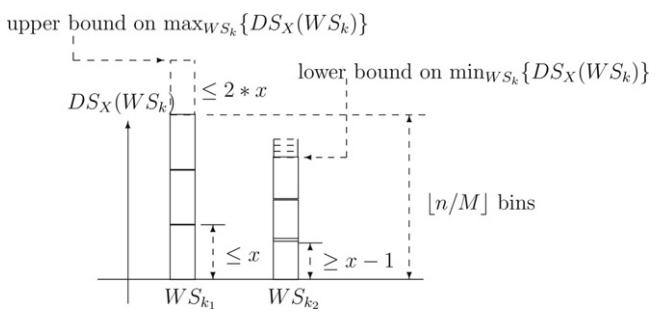


Fig. 11. Effects of bin capacity selection on storage balancing.

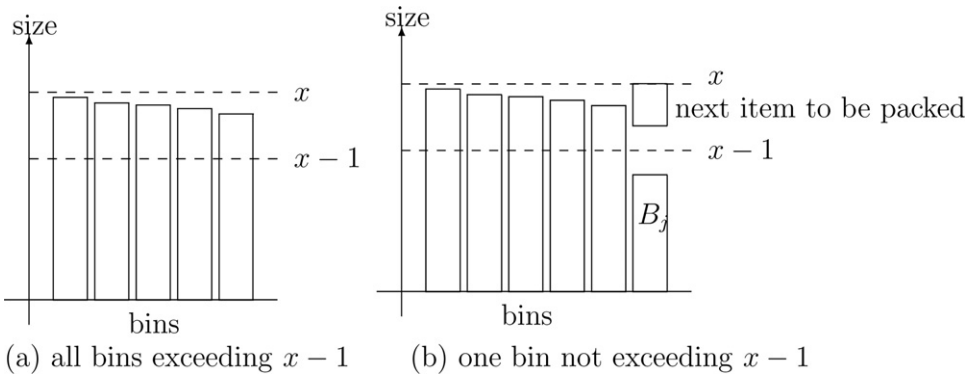


Fig. 12. (a and b) Packing an item to bins with enlarged bin capacity.

Similar to Lemma 2, following lemma shows the upper bound on the number of bins generated. (S is the total size of all items.)

Lemma 4. *With bin capacity $x > 1$, the number of bins n generated by bin packing is bounded,*

$$n \le \frac{S}{x-1} + 1. \tag{16}$$

Proof. According to Lemma 3, there are at least $n-1$ bins with sizes exceeding $x-1$, and the total data size S exceeds the total size of these $n-1$ bins,

$$S \ge (x-1) \times (n-1).$$

Eq. (16) is obtained immediately. □

The storage requirement of a workstation can then be written as a function of bin capacity x, stated as follows. (M is the number of workstations.)

Lemma 5. *With bin capacity $x > 1$, the proposed algorithm generates an allocation X in which*

$$DS_X(WS_k) \le \frac{S}{M} \times \left(1 + \frac{1}{x-1}\right) + 3x \tag{17}$$

for any workstation WS_k.

Proof. In the output of the proposed algorithm, each workstation WS_k contains at most $\lceil n/M \rceil + 1$ bins with the use capacity not exceeding x for all bins.

$$DS_X(WS_k) \le \left(\left\lceil \frac{n}{M} \right\rceil + 1\right) \times x.$$

Lemma 4 gives an upper bound on n and Eq. (17) is obtained immediately. □

The bin capacity is selected to minimize the storage requirement function. The **storage requirement function** $f(x)$ indicates the required storage of a workstation if bin capacity is set to be x.

$$f(x) \equiv \frac{S}{M} \times \left(1 + \frac{1}{x-1}\right) + 3x. \tag{18}$$

The storage requirement function on the x-y plane is shown in Fig. 13, which reflects the tradeoff in selecting the bin capacity. Solving the differential equation $f'(x) = 0$, we obtain the **optimal bin capacity** x_0 to minimize $f(x)$:

$$x_0 = 1 + \sqrt{\frac{S}{3 \times M}}. \tag{19}$$

And the most efficient required storage capacity for a workstation is obtained:

$$f(x_0) = \frac{S}{M} + 2\sqrt{3} \times \sqrt{\frac{S}{M}} + 3. \tag{20}$$

Theorem 3 (Workstation storage requirement for bin capacity ¿1). *By selecting bin capacity $x = 1 + \sqrt{S/(3 \times M)}$, the proposed algorithm generates an allocation X in which*

$$DS_X(WS_k) \le \frac{S}{M} + 2\sqrt{3} \times \sqrt{\frac{S}{M}} + 3 \tag{21}$$

for any workstation WS_k.

Proof. This is the conclusion of previous discussion. □

The theorem indicates the optimization quality of the proposed algorithm in storage balancing.

4.3. Summary of proposed algorithm

We summarize previous discussion to derive a complete algorithm and analyze the complexity and asymptotic behavior of the algorithm.

The proposed data allocation algorithm, *LSB_Alloc* (Load and Storage Balanced Allocation), is shown in Fig. 14. The algorithm determines the bin capacity as discussed in Section 4.2. By comparing Eq. (14) and Eq. (21), whether to enlarge the bin capacity or not is best determined by S/M, where S is the total data size and M is the number of workstations. (Note that the results of the two equations Eqs. (14) and (21) equals at $S/M = 12$.) Properties of the output are proved in previous sections.

The complexity of the data allocation algorithm is as follows. The time complexity of best-fit bin packing is $O(N^2)$ (Horowitz et al., 1996). The time complexity of the MMPacking to allocate n ($\le N$) bins is $O(n+M)$ (Serpanos et al., 1998). Hence the time complexity of the proposed algorithm is $O(N^2 + M + n)$. To implement the algorithm, an $O(n \times M)$ space is required to store the result of MMPacking Y. The final result X can be implemented as a mapping

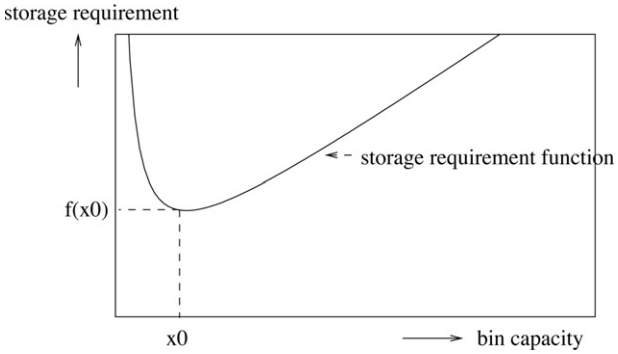


Fig. 13. Capacity function for selecting bin capacity.

Algorithm $LSB\_PostingFilePartition(PF_{seq}, TP, WS, PPF)$

- Input:
  - $PF_{seq} = \{L_0, L_1, ..., L_{n-1}\}$: a posting file for sequential query processing
    - L_t: posting list of term t
  - $TP = \{p_0, p_1, ..., p_{n-1}\}$: term popularities
    - p_t: probability that term t appears in a query
  - $WS = \{WS_0, WS_1, ..., WS_{M-1}\}$: a set of workstations
- Output:
  - $PPF = \{LPF_0, LPF_1, ...LPF_{M-1}\}$: the partitioned posting file
    - $LPF_k = \{L_0(WS_k), L_1(WS_k), ..., L_{n-1}(WS_k)\}$: local posting file at WS_k, consisting of a local posting list $L_t(WS_k)$ for each term t
- Method:

(1) /* scan PF_{seq} to assign parameters to each item */
  (1.1) **for** each document ID i **do** initialize I_i: $Load(I_i) \leftarrow 0$ and $postings(I_i) \leftarrow 0$
  (1.2) $MaxItemSize \leftarrow 0$
  (1.3) **for** each posting list $L_t \in PF_{seq}$ **do**
    for each document ID $i \in L_t$ **do**
    (1.3.1) $Load(I_i) \leftarrow Load(I_i) + p_t$
    (1.3.2) $postings(I_i) \leftarrow postings(I_i) + 1$
    (1.3.3) **if** $MaxItemSize < postings(I_i)$ **then** $MaxItemSize \leftarrow postings(I_i)$
  (1.4) **for** each document ID i **do** $s_i \leftarrow \frac{postings(I_i)}{MaxItemSize}$
(2) perform $LSB\_Alloc(I, WS, X)$
(3) /* rescan PF_{seq} to generate the partitioned posting file from X */
  (3.1) **for** each $WS_k \in WS$ **do**
    for each term t **do** $L_t(WS_k) \leftarrow \phi$
  (3.2) **for** each posting list $L_t \in PF_{seq}$ **do**
    for each document ID $i \in L_t$ **do**
      append document ID i to $L_t(WS_k)$ if $X_{ik} = 1$

Fig. 14. Proposed load and storage balanced data allocation algorithm, *LSB_Alloc*.

table with $O(N)$ space. Hence the space complexity of the algorithm is $O(n \times M + N)$.

The algorithm is asymptotically 1-optimal on storage balancing. Let J be an instance (for given data items I and workstations WS) for the optimization problem. Eq. (3) defines the cost of a solution for J. The notation $OPT(J)$ denotes the cost of the optimal solution and $F(J)$ denotes the cost of the solution found by the proposed algorithm. It is clear that $OPT(J) \geq S/M$. According to Theorem 3, the ratio $F(J)/OPT(J)$ is bounded as follows.

$$\frac{F(J)}{OPT(J)} \leq 1 + \frac{2\sqrt{3}}{\sqrt{S/M}} + \frac{3}{S/M}. \quad (22)$$

Fig. 15 depicts the curve of Eq. (22). The ratio $F(J)/OPT(J)$ approaches one when S/M exceeds certain threshold. Section 7 shows that very near optimal storage balancing is achieved for real-world applications.

5. Parallel information retrieval

We come back to information retrieval problem. The work here is to apply the proposed data allocation algorithm *LSB_Alloc* for posting file partitioning. We first specify what a data item is. We follow the partition-by-document-ID principle to partition the posting file. The principle states that an item for data allocation is

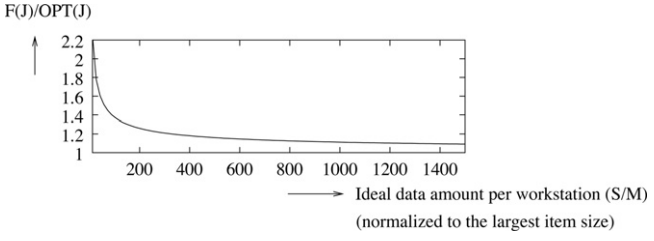


Fig. 15. Storage balancing property of proposed algorithm.

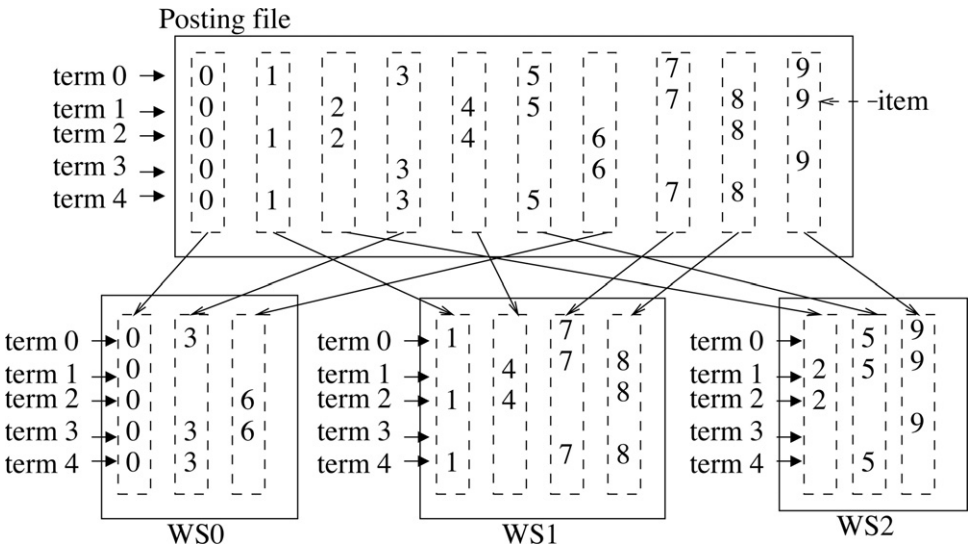


Fig. 16. The partition-by-document-ID principle.

the set of all postings referring to the same document ID. This section describes how a query is processed in parallel following the principle. With this principle, a query is processed in parallel without having to transferring postings between workstations. Time complexity of parallel query processing will be analyzed. Based on the analysis, Section 6 formulates posting file partitioning as the data allocation problem and proposes the posting file partitioning algorithm.

The partition-by-document-ID principle is illustrated in Fig. 16. The principle dictates that all postings referring to the same document ID be allocated on the same workstation. Each workstation covers an exclusive set of document IDs. In parallel query processing, workstation WS_k is responsible for providing answers only from document IDs covered by it. For example, for the query (term 1 < AND> term 2), WS_1 provides answers $\{4, 8\}$. Checking whether a document ID d matches a query or not requires only postings referring to document ID d, which are all in the same workstation. For the above example, checking whether document 4 contains both term 1 and term 2 requires only the local data in WS_1. Following the principle, a query is processed independently and in parallel by all workstations.

This section describes how a query is processed in parallel. Section 5.1 deals with the set theory and the time complexity of parallel query processing. Section 5.2 deals with implementation issues on cluster of workstations.

5.1. The theory

The partitioned posting file is formalized as follows. To partition a posting file is to map document IDs to workstations. Let L_t be the posting list of term t, and D_k be the set of document IDs mapped to WS_k. The notation $L_t(WS_k)$ denotes the set of document IDs in L_t which are mapped to WS_k.

$$L_t(WS_k) = L_t \cap D_k. \tag{23}$$

The **local posting list** of term t in WS_k is the set of document IDs in $L_t(WS_k)$ stored in increasing order. The **local posting file** in WS_k is the set of local posting lists for all terms t. For the example in Fig. 16, local posting files for all workstations are shown in Fig. 17.

Parallel query processing works as follows. For a given query q, the parallel query processing is to compute the answer list ANS_q in parallel. Each workstation WS_k is responsible for computing its own **partial answer list** $ANS_q(WS_k)$:

$$ANS_q(WS_k) = ANS_q \cap D_k. \tag{24}$$

The set $ANS_q(WS_k)$ is the set of all document IDs matching query q and mapped to WS_k. The union of all partial answer lists from all workstations is hence the complete answer list,

$$Ans_q = \bigcup_{WS_k} Ans_q(WS_k). \tag{25}$$

The following theorem states the set operation to compute a partial answer list.

Theorem 4 (Computation of partial answer list). *The partial answer list, $ANS_q(WS_k)$, can be represented in set operations on local posting lists of queried terms in WS_k.*

Proof. We prove this theorem by induction on the number of Boolean operators in the given query q. The induction hypothesis is the theorem itself.

The basis, when q contains only one Boolean operator, is as follows. Query q is either "term i < AND> term j" or "term i < OR> term j". Consider the case that q is "term i < AND> term j". The partial answer list at WS_k is

$$ANS_q(WS_k) = (L_i \cap L_j) \cap D_k = (L_i \cap D_k) \cap (L_j \cap D_k) = L_i(WS_k) \cap L_j(WS_k).$$

This rewrites $ANS_q(WS_k)$ with set operations on local posting lists in WS_k. The case that q is "term i < OR> term j" is similar and is omitted.

Suppose the theorem holds when the number of Boolean operators in q is less than n and we prove that the theorem also holds when q contains n Boolean operators. The query q is either "(q_1) < AND> (q_2)" or "(q_1) < OR> (q_2)" where q_1 and q_2 are queries containing no more than $n - 1$ Boolean operators. Consider the case that q

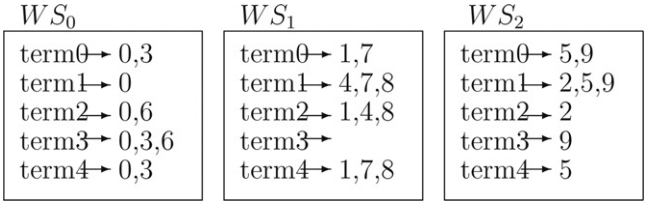


Fig. 17. Local posting files in above example.

is "(q_1) < AND> (q_2)". The partial answer list at WS_k is

$$ANS_q(WS_k) = (ANS_{q_1} \cap ANS_{q_2}) \cap D_k = (ANS_{q_1} \cap D_k) \cap (ANS_{q_2} \cap D_k).$$

The above equation can be written as

$$ANS_q(WS_k) = ANS_{q_1}(WS_k) \cap ANS_{q_2}(WS_k).$$

The induction hypothesis states that $ANS_{q_1}(WS_k)$ and $ANS_{q_2}(WS_k)$ can be represented in set operations on local posting lists of queried terms in WS_k, and hence so is $ANS_q(WS_k)$. The case that q is "(q_1) < OR> (q_2)" is similar and is omitted. This theorem is hence proved by induction. □

Theorem 4 states an efficient way to compute a partial answer list. To compute $ANS_q(WS_k)$, WS_k only has to perform basic set operations on its local posting lists of queried terms, without examining all document IDs mapped to it. An example is as follows. Consider the partitioned posting file in Fig. 16, and let query q be "(term0 < AND> term3) < OR> term4". Local posting lists of term0, term3, and term4 in WS_2 are {5, 9}, {9}, and {5}, respectively. The series of set operations to compute the partial answer list at WS_2 is as follows:

$$ANS_q(WS_2) = (\{5, 9\} \cap \{9\}) \cup \{5\} = \{5, 9\}.$$

Note that no postings referring to document ID 2 is examined.

The time complexity of computing a partial answer list is as follows. Any set operation algorithm operating on sorted lists can be used. We use the list merging algorithm (Salton, 1989) to perform set operations. Let $f_{t_i}^{(k)}$ be the length of the local posting list of the i th queried term t_i in WS_k, and m be the number of queried terms. The following corollary states the time complexity.

Corollary 4. *With list merging* (Salton, 1989), *the time complexity to compute* $ANS_q(WS_k)$ is $O(f_{t_1}^{(k)} + f_{t_2}^{(k)} + \cdots + f_{t_m}^{(k)})$.

5.2. Implementation on cluster of workstations

This section describes the flow of processing a query on a cluster of workstations, starting from receiving a user query to the completion of the answer list. While Boolean query processing is the major concern here, the proposed scheme can also be extended to deal with ranking with the addition of parallel sorting.

The flow of computing the answer list for a query is as follows. Fig. 18 shows the system overview of a clustered information retrieval system. The parallel flow to compute the answer list for such a cluster is shown in Fig. 19. A specific workstation, called the gateway, is dedicated for receiving user queries and performing the index file search. The gateway searches the index file shown in Fig. 20(a), and substitutes a term ID for each term in the query. All the other workstations are called the backend workstations. Records of frequently used terms are often stored in random access memory so that the average index search time will not scale with the size of the keyword collection. The query is then broadcasted to all back-end workstations to compute the answer list in parallel. Each workstation stores an index array of pointers to local posting lists, as shown in Fig. 20(b). Upon receiving a broadcasted query, the workstation retrieves local posting lists and computes its own partial answer list. The partial answer list is buffered locally, and the number of document IDs found is sent back to the gateway.

The remaining work is for the gateway to reply answers to the user page by page. A page contains the number of answers to the query, and a page full of titles of matched documents. The number of answers is useful for a user to determine whether a different query should be requested: for example, when the number of answers is large, a user may decide to discard the query results and give a more specific query. The gateway accumulates number of answers found by each back-end workstation to obtain total number of answers. The first page is then generated and delivered to the user. Parallelization of query processing reduces the time to deliver the first page, in which the total number of answers must be contained. Remaining pages are generated and delivered upon user demands. To generate a page, the gateway polls some back-end workstation(s) to get answers just enough to fill a page. Since a user may not request all of the results to a query, the answers distributed on multiple workstations need not be collected at once.

Recent progress on parallel sorting provides efficient ways to rank answers on multiple workstations. Let r be the number of answers to be presented on a page. Each workstation scores and selects the top r answers within its partial answer list independently. The top r answers in the complete answer list is obtained by parallel sorting (Kumar, 1994) of all workstations' top r answers. With architectural support, Patterson's group shows that more than 1 G integers can be sorted in 2.41 s using 64 workstations (Arpaci-Dusseau et al., 1997, 1998). The time to rank answers for a page is small since r is small and does not scale with the collection size.

6. Posting file partitioning algorithm

We are now ready to partition the posting file using the data allocation algorithm. The input includes

- posting file PF_{seq} good only for sequential processing,
- popularities of keyword terms p_t for each term t, and
- a set of workstations $WS = \{WS_0, WS_1, \ldots, WS_{M-1}\}$.

The output is a partitioned form of PF_{seq} to be distributed on WS, following the partition-by-document-ID principle. Mean query processing time in parallel processing of the partitioned posting file is estimated according to popularities of keyword terms. The objective is to minimize the storage requirement per workstation subject to the constraint: the mean query processing time of a workstation is at most one document processing time more than the ideal value. We first formulate posting file partitioning as the data allocation problem defined in Section 3.1. The proposed algorithm *LSB_Alloc* is then applied to generate a partitioned posting file.

6.1. Formulating as data allocation problem

Posting file partitioning can be formulated as the data allocation problem defined in Section 3.1. Three rules are given to specify (1) an item, (2) size of an item, and (3) load of an item. The key issue is to define item loads such that the mean query processing time of a workstation can be calculated by accumulating loads of allocated items. We establish a probability model to define item loads.

The following rule specifies what an item is. This rule is the partition-by-document-ID principle described in Section 5.

Rule 1 *An item* I_i *to be allocated by Algorithm LSB_Alloc is the set of all postings referring to doc. ID* i.

With the rule, a partitioned posting file is generated as follows. The algorithm *LSB_Alloc* generates an allocation matrix X, indicating the mapping of document IDs (items) to workstations. For the allocation X, local posting list of term t at workstation WS_k, denoted $L_t(WS_k)$, is as follows.

$$L_t(WS_k) = \{\text{doc. ID}i | i \in L_t \text{and} X_{ik} = 1\}. \quad (26)$$

The local posting file at WS_k is the set of local posting lists for all term t.

The following rule specifies the size of an item.

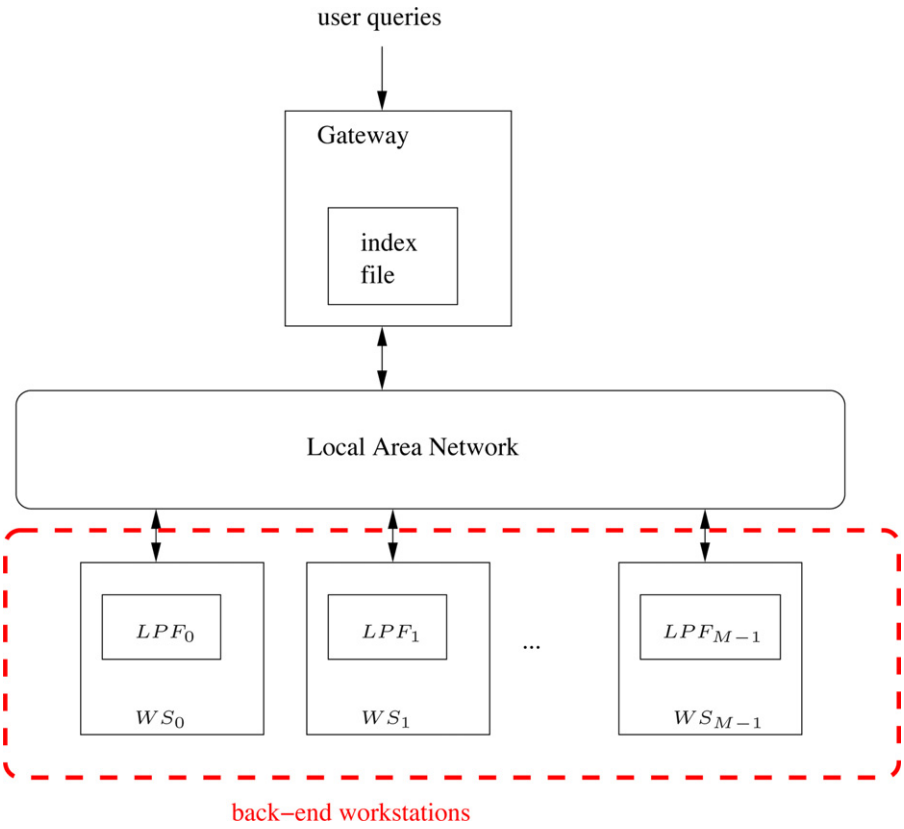


Fig. 18. System overview of a clustered information retrieval system.

Rule 2 *Data size s_i of item I_i is normalized and defined as follows:*

$$s_i = \frac{\text{number of postings referring to doc. ID } i}{\max_{\text{doc. ID } j}\{\text{number of postings referring to doc. ID } j\}}. \quad (27)$$

Storage requirement of a workstation is calculated as follows. The data size allocated on workstation WS_k, denoted $DS_X(WS_k)$ defined in Eq. (2), indicates (normalized) amount of postings allocated on WS_k. The space, in bytes, occupied by an item I_i is [(bytes per posting)× (number of postings in I_i)]. For workstation WS_k, the required storage space in bytes is [$DS_X(WS_k)$ × (space occupied by the largest item)].

Mean query processing time is estimated by the following probability model. Let TQ_t be a random Boolean variable representing whether term t appears in a query: $TQ_t = 1$ if term t appears in a query and $TQ_t = 0$ otherwise. The **term popularity** p_t of a term t is the probability that a query contains the term t. That is, $p_t = Pr\{TQ_t = 1\}$. The expected value of TQ_t is thus

$$E[TQ_t] = 1 \times Pr\{TQ_t = 1\} + 0 \times Pr\{TQ_t = 0\} = p_t. \quad (28)$$

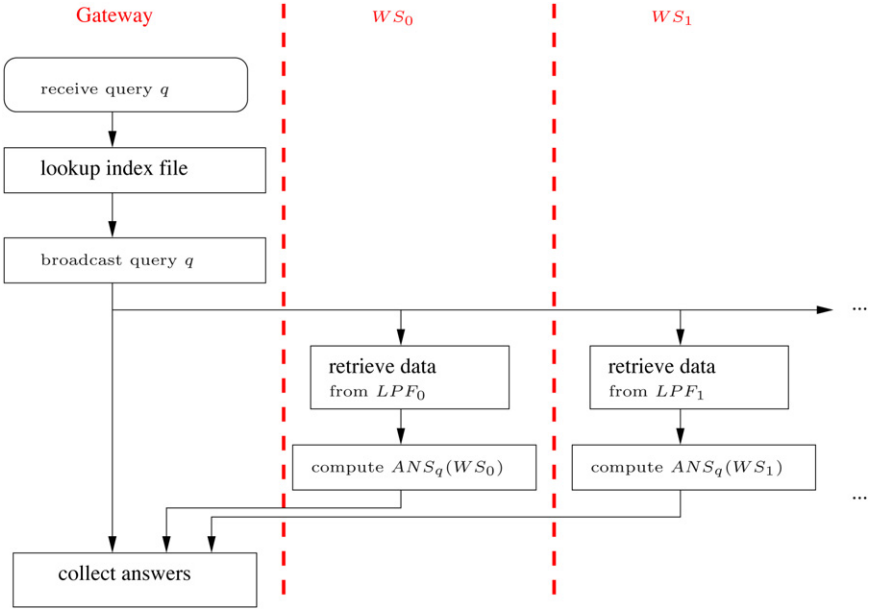


Fig. 19. Flow of parallel query processing.

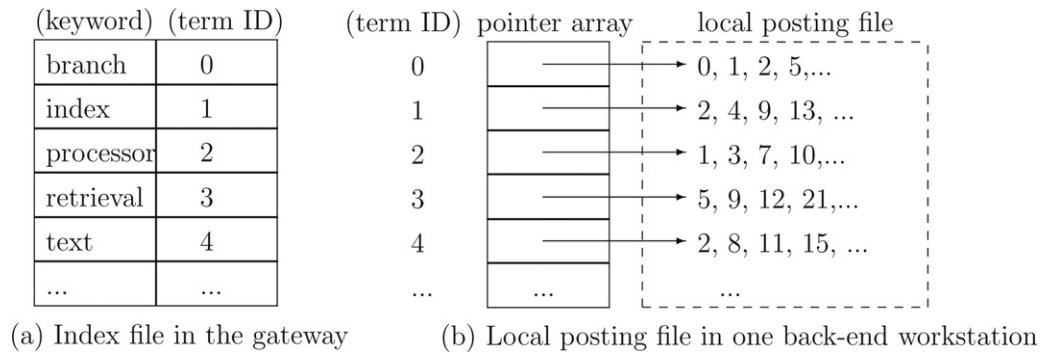


Fig. 20. (a and b) Partitioned inverted file on cluster of workstations.

Let $f_t^{(k)}$ be the length of the local posting list of term t in workstation WS_k. Corollary 4 states that the query processing time is proportional to amount of postings to be processed. With allocation X, the query processing time at WS_k, denoted $QPT_X(WS_k)$, is as follows:

$$QPT_X(WS_k) = \sum_{\text{term}t} TQ_t \times f_t^{(k)}, \tag{29}$$

where time quantity is normalized such that

- one unit of time is the average time to process a posting.

The mean query processing time of WS_k is the expected value of $QPT_X(WS_k)$.

$$MQPT_X(WS_k) = E[QPT_X(WS_k)]. \tag{30}$$

Similarly, the sequential query processing time, denoted QPT_{seq}, and the mean sequential query processing time, denoted $MQPT_{seq}$, are as follows:

$$QPT_{seq} = \sum_{\text{term}t} TQ_t \times f_t, \tag{31}$$

$$MQPT_{seq} = E[QPT_{seq}]. \tag{32}$$

(Notation f_t stands for the length of the posting list of term t.)

Item loads are defined to indicate mean query processing time, as stated in the following rule and theorems. Let L_t be the posting list of term t. The rule to define item loads is as follows.

Rule 3 *The load of the item I_i is as follows:*

$$Load(I_i) = \sum_{\text{term}t:i\in L_t} E[TQ_t] = \sum_{\text{term}t:i\in L_t} p_t. \tag{33}$$

The load of an item I_i can be calculated by accumulating corresponding term popularities for all postings in I_i. Consider the example in Fig. 16. There are three postings in the item corresponding to document ID 7: postings corresponds to terms 0, 1, and 4. The load is thus $Load(I_7) = p_0 + p_1 + p_4$. The load of I_i is the aggregated mean query processing time imposed by I_i. This is stated in the following theorems.

Theorem 5 (Mean query processing time in parallel processing). *Mean query processing time of a workstation WS_k is the summed load of all items allocated on WS_k.*

$$MQPT_X(WS_k) = \sum_{I_i:X_{ik}=1} Load(I_i) = Load_X(WS_k). \tag{34}$$

Proof. Eq. (34) is derived by rewriting $QPT_X(WS_k)$ as accumulating TQ_t s corresponding to all postings. Refer againto Fig. 16. $QPT_X(WS_k)$ can be calculated by scanning the local posting file row by row. Each time a posting is found, the corresponding TQ_t is added to the current sum. This rewrites Eq. (29) to be

$$QPT_X(WS_k) = \sum_{\text{term}t} \left(\sum_{I_i:X_{ik}=1 \text{and} i\in L_t} TQ_t \right)$$

(where L_t is the posting list of term t). Scanning the local posting file column by column also yields the same result and the above equation is equivalent to:

$$QPT_X(WS_k) = \sum_{I_i:X_{ik}=1} \left(\sum_{\text{term}t:i\in L_t} TQ_t \right).$$

The mean query processing time is thus:

$$MQPT_X(WS_k) = \sum_{I_i:X_{ik}=1} \left(\sum_{\text{term}t:i\in L_t} E[TQ_t] \right).$$

By observing Eq. (33) and the above equation, Eq. (34) is obtained. □

Theorem 6 (Mean query processing time in sequential processing). *Mean query processing time in sequential processing is the total load of all items.*

$$MQPT_{seq} = \sum_{I_i} Load(I_i) = L. \tag{35}$$

Proof. This is similar to the proof of Theorem 5. □

With these three rules, Algorithm *LSB_Alloc* can be applied to generate a partitioned posting file with the following properties. The three rules specify inputs to Algorithm *LSB_Alloc*. Algorithm *LSB_Alloc* then generates an allocation X, and the partitioned posting file is generated from X according to Eq. (26). The objective of posting file partitioning is to balance amount of postings allocated on workstations subject to a limited difference to ideal mean query processing time. Storage requirement is indicated by Theorems 2 and 3. Mean query processing time in parallel processing is stated in the following Corollary, which is a direct consequence of Theorems 1, 5, and 6.

Corollary 5. *Applying AlgorithmLSB_Alloc generates a partitioned posting file such that*

$$MQPT_X(WS_k) \le \frac{MQPT_{seq}}{M} + \max_{I_i}\{Load(I_i)\} \tag{36}$$

for any workstation WS_k.

Algorithm $LSB\_PostingFilePartition(PF_{seq}, TP, WS, PPF)$

- Input:
  - $PF_{seq} = \{L_0, L_1, ..., L_{n-1}\}$: a posting file for sequential query processing
    - L_t: posting list of term t
  - $TP = \{p_0, p_1, ..., p_{n-1}\}$: term popularities
    - p_t: probability that term t appears in a query
  - $WS = \{WS_0, WS_1, ..., WS_{M-1}\}$: a set of workstations
- Output:
  - $PPF = \{LPF_0, LPF_1, ...LPF_{M-1}\}$: the partitioned posting file
    - $LPF_k = \{L_0(WS_k), L_1(WS_k), ..., L_{n-1}(WS_k)\}$: local posting file at WS_k, consisting of a local posting list $L_t(WS_k)$ for each term t
- Method:

(1) /* scan PF_{seq} to assign parameters to each item */
  (1.1) **for** each document ID i **do** initialize I_i: $Load(I_i) \leftarrow 0$ and $postings(I_i) \leftarrow 0$
  (1.2) $MaxItemSize \leftarrow 0$
  (1.3) **for** each posting list $L_t \in PF_{seq}$ **do**
    for each document ID $i \in L_t$ **do**
    (1.3.1) $Load(I_i) \leftarrow Load(I_i) + p_t$
    (1.3.2) $postings(I_i) \leftarrow postings(I_i) + 1$
    (1.3.3) **if** $MaxItemSize < postings(I_i)$ **then** $MaxItemSize \leftarrow postings(I_i)$
  (1.4) **for** each document ID i **do** $s_i \leftarrow \frac{postings(I_i)}{MaxItemSize}$
(2) perform $LSB\_Alloc(I, WS, X)$
(3) /* rescan PF_{seq} to generate the partitioned posting file from X */
  (3.1) **for** each $WS_k \in WS$ **do**
    for each term t **do** $L_t(WS_k) \leftarrow \phi$
  (3.2) **for** each posting list $L_t \in PF_{seq}$ **do**
    for each document ID $i \in L_t$ **do**
      append document ID i to $L_t(WS_k)$ if $X_{ik} = 1$

Fig. 21. Algorithm for generating partitioned posting file, *LSB_PostingFilePartition*.

Corollary 5 states that the mean query processing time of a workstation is at most the ideal value, $MQPT_{seq}/M$, plus the effect of a single document.

6.2. Generation of partitioned posting file

Fig. 21 shows the algorithm to generate the partitioned posting file. The first step scans the input posting file to assign parameters to items. Algorithm *LSB_Alloc* is then invoked to obtain allocation matrix *X*. Finally, the input posting file is scanned again to generate the partitioned posting file from *X*.

The complexity of generating a partitioned posting file is as follows. Let N be the number of documents, M be the number of workstations, n be the number of bins generated by bin packing, and f be the number of postings in the input posting file. The time complexity is $O(N^2 + M + n + f)$, in which $O(N^2 + M + n)$ is spent on the algorithm *LSB_Alloc* and $O(f)$ is spent on scanning the input posting file twice. The space complexity is $O(n \times M + N + f)$, in which $O(f)$ space is used to store the input and generated posting file and $O(n \times M + N)$ space for algorithm *LSB_Alloc*.

7. Application: quantitative method for workstation cluster design

With the proposed posting file partitioning algorithm, we are ready to show a quantitative method to design a parallel information retrieval system systematically. The quantitative method determines number of workstations and storage capacity per workstation of a cluster. The objective of the quantitative method is to minimize the hardware cost to satisfy a given throughput requirement. Load balancing reduces the number of workstations to satisfy a given throughput requirement. Storage balancing reduces storage requirement of all workstations. We show the usefulness of our work on real-world applications with TREC document collection (Hardman, 1992).

7.1. Cluster configuration problem

The concerned problem is to determine the workstation cluster configuration according to statistical data. The input includes a posting file for sequential query processing and the following parameters obtained from profiling:

- term popularity p_t for each term t,
- average time per posting in sequential query processing, and
- throughput requirement λ.

The output, workstation cluster configuration, should specify

- number of workstations M in the cluster,
- storage capacity CAP per workstation, and
- data allocation X to generate the partitioned posting file for the M workstations.

Hardware cost of the cluster is [$M \times$ (cost per workstation)]. The cost per workstation is a function of the storage capacity CAP. The objective is to minimize the hardware cost subject to the following constraints:

- throughput of the cluster $\geq \lambda$, and
- amount of data allocated on a workstation $\leq$ storage capacity CAP,

$DS_X(WS_k) \leq CAP$ for any WS_k.

The following throughput requirement is assumed.

Assumption 1. The throughput requirement λ satisfies

$$\lambda < \frac{1}{\max_{I_i}\{Load(I_i)\}}. \tag{37}$$

Recall that the load of an item I_i is the mean query processing time of document i. And the time unit is normalized such that one unit of time is average processing time per posting. Assumption 1 ensures the existence of a solution to the cluster configuration problem (see the next subsection).

7.2. Cluster configuration procedure

Workstation cluster configuration is calculated according to load and storage balancing properties of proposed data allocation algorithm. The key issue is to relate throughput requirement to the load balancing property.

Load balancing property determines the throughput capability of a cluster. Theorem 5 states that load allocated on a workstation is the mean query processing time of the workstation. Load balancing is to minimize the maximum amount of load allocated on a single workstation. That is, to minimize

$$\max_{WS_k}\{Load_X(WS_k)\} = \max_{WS_k}\{MQPT_X(WS_k)\}.$$

With a query arrival rate of λ, the mean query processing time of each workstation should not exceed the average time interval between two arrived queries. That is,

$$MQPT_X(WS_k) < \frac{1}{\lambda} \text{ for any } WS_k$$

or equivalently,

$$\max_{WS_k}\{MQPT_X(WS_k)\} < \frac{1}{\lambda}.$$

With fixed number of workstations M, load balancing is to maximize the throughput capability λ. Whereas if throughput requirement is given and M is to be determined, load balancing is to minimize M that meets the throughput requirement.

Performance limitation exists for parallel information retrieval. Following the partition-by-document-ID principle, no parallel information retrieval system can achieve throughput $\lambda > 1/\max_{I_i}\{Load(I_i)\}$. The throughput limit is derived as follows. Let I_j be the item with maximum load among all items. Suppose the throughput requirement λ exceeds the inverse of the maximum item load.

$$\lambda > \frac{1}{\max_{I_i}\{Load(I_i)\}} = \frac{1}{Load(I_j)}.$$

Let WS_k be the workstation that I_j is assigned to. The mean query processing time of WS_k is at least the load of I_j, which exceeds $1/\lambda$.

$$MQPT_X(WS_k) = Load_X(WS_k) \geq Load(I_j) > \frac{1}{\lambda}.$$

The workstation WS_k is overwhelmed by the query arrival rate λ. According to the throughput limitation, Assumption 1 ensures the existence of a solution to the cluster configuration problem.

Cluster configuration is calculated as follows. According to Corollary 5, the required number of workstations M to achieve throughput requirement λ is:

$$M = \left\lceil \frac{MQPT_{seq}}{(1/\lambda) - \max_{I_i}\{Load(I_i)\}} \right\rceil. \tag{38}$$

Note that Assumption 1 ensures that $(1/\lambda) - \max_{I_i}\{Load(I_i)\} > 0$. Let S be the total data size. Storage requirement CAP is determined according to Theorem 3:

$$CAP \geq \frac{S}{M} + 2\sqrt{3} \times \sqrt{\frac{S}{M}} + 3. \tag{39}$$

Recall that quantities in Eq. (39) is normalized with the factor that one unit of space is the space occupied by the largest item. The next subsection justifies that S/M exceeds 12 in real world applications.

8. Evaluation of proposed posting file partitioning algorithm

This section presents our evaluation on the proposed posting file partitioning algorithm. The evaluation is to show the effectiveness of the proposed algorithm on real-world large-scale document collection. We use TREC/Blogs08 (Macdonald et al., 2010) as the document collection and AOL query log as sample queries for this evaluation. The evaluation result shows that, with the proposed algorithm, one may design a clustered information retrieval system with simple quantitative method and still approximates optimal throughput and storage cost.

8.1. Evaluation method on load and storage balancing effect

The evaluation method is as follows. The objective of the evaluation is to show how throughput and storage cost scales with cluster size for real-world document collection. An advantage of our proposed algorithm is that we have analytical properties on load and storage balancing properties been proved. We thus profile the inverted file built upon the document collection to get statistics data. Throughput and storage cost of a clustered information retrieval system is thus calculated from the analytical properties with the statistics data.

Fig. 22 shows the evaluation flow to collect required data. Utility programs built for collecting data are (1) inverted file builder, (2) query-log profiler, (3) item profiler, and (4) query evaluator. The evaluation flow is as follows.

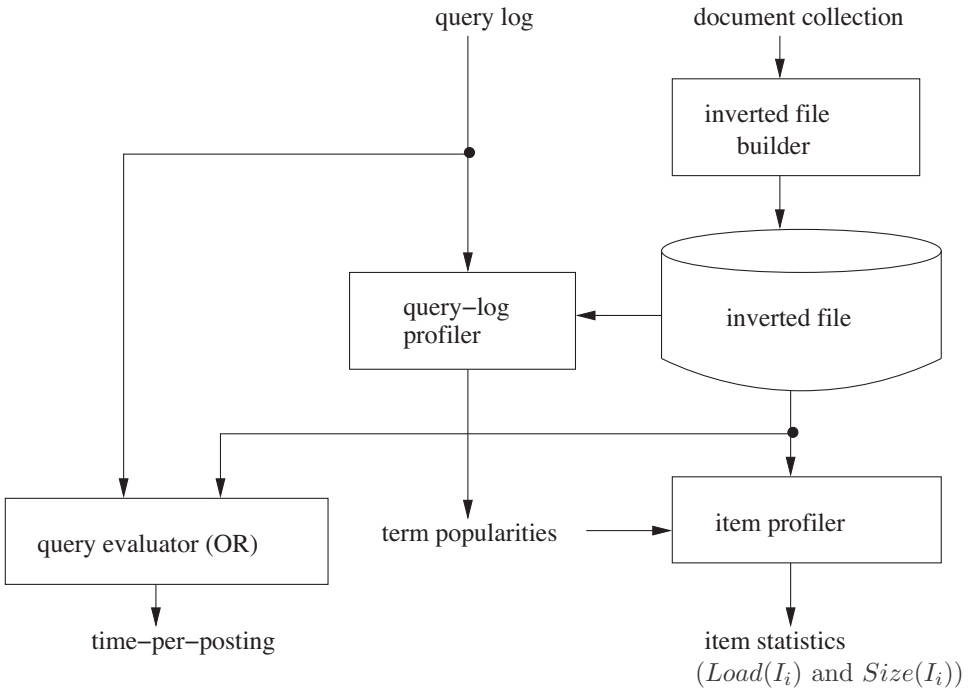


Fig. 22. Overview on the experiment system.

- Stage 1: We build an inverted file over the test document collection. The MG package (Witten et al., 1999) is applied to build the inverted file. The inverted file built by MG is then dumped and converted to our internal format for the convenience of the evaluation.
- Stage 2: The query-log profiler reads all queries in the AOL query log and annotates appearance counts for all indexed keywords in the inverted file. This generates term popularities, probability that a term appears in a query, for all keyword terms.
- Stage 3: The item profiler takes the term popularities and reads the whole posting file to get statistics data for all data items. An item is the set of postings referring to the same document ID. This stage generates loads and data sizes for all data items.
- Stage 4: The query evaluator performs query processing for all queries in the test query log. An OR operation is performed on all keyword terms in a query. Besides the query results, the processing time on the test computer is also recorded. With this utility, we got average time-per-posting for query processing.

The throughput of a clustered information retrieval system constructed by the proposed algorithm is calculated as follows. We re-write formulas in Sections 6 and 7 to get the throughput equation in unit of queries-per-second. With M workstations, the throughput $\lambda(M)$ is guaranteed to meet the following lower-bound:

$$\lambda(M) \geq \frac{1}{TPP \times ((L/M) + \max_{I_i}\{Load(I_i)\})}, \tag{40}$$

where TPP is the average time-per-posting and L is the total load of all data items. The ratio-to-ideal on throughput aspect is thus as follows:

$$\frac{\lambda(M)}{\lambda_{ideal}(M)} \geq \frac{1}{1 + \max_{I_i}\{Load(I_i)\}/(L/M)}, \tag{41}$$

where the ideal throughput $\lambda_{ideal}(M)$ occurs when each workstation is allocated balanced load L/M.

With the proposed algorithm, storage requirement per workstation is also calculated from re-writing formulas in Sections 6 and 7.

For a cluster of M workstations, the required storage space per workstation in bytes is obtained from the following equation:

$$SC(M) \leq MIS \times \left(\frac{S}{M} + 2\sqrt{3}\sqrt{\frac{S}{M}} + 3\right), \tag{42}$$

where MIS is the maximum item size in bytes and S is the total data size normalized such that the maximum item size is 1. The ratio-to-ideal on the aspect of storage cost is calculated from the following equation:

$$\frac{\max_{WS_k}\{DS_X(WS_k)\}}{S/M} \leq 1 + \frac{2\sqrt{3}}{\sqrt{S/M}} + \frac{3}{S/M}. \tag{43}$$

Note that all parameters required to calculate throughput and storage requirement are obtained from the evaluation system in Fig. 22.

8.2. Evaluation results and analysis

We present the evaluation results as a case study. Consider the case of setting up a clustered information retrieval system over TREC/Blogs08 document collection (Macdonald et al., 2010) with profiling information provided from AOL query log. Table 1 shows basic statistics data over TREC/Blogs08. The system manager may setup the cluster with the following steps:

- Step 1: Find a range on amount of workstations to fit storage capacity per workstation. Thestorage requirement for a worksta-

Table 1
Basic data of TREC/Blogs08 document collection.

Total documents	624,470
Total postings	4,545,314,247
Maximum amount of postings for a document (item)	15,979
Total load	513,258.210938
Maximum load of an item	1.033949

Table 2
Statistics data for performance.

Parameter	Value	Comments
Average time per posting *TPP*	0.009701 μs	Operating with RAM
Total load *L*	513,258.210938	Postings a query accessed
Maximum item load $max_{I_i}\{Load(I_i)\}$	1.033949	Postings referred to a document
Total data size *S*	284,455	Normalized to maximum item size
Maximum item size	1.00	Normalized for calculation

tion can be calculated from Eq. (42). Ratio-to-ideal on the aspect of storage cost is given by Eq. (43).

- Step 2: Observe the query processing throughput for the interested cluster size addressed in Step 1. The query processing throughput is indicated by Eq. (40). The ratio-to-ideal on throughput aspect is given by Eq. (41).
- Step 3: Make the final decision on cluster size from observations in Step 1 and 2.

Table 2 shows collected parameters over TREC/Blogs08 for calculating throughput and storage cost with the mentioned equations. We thus observe the cluster configuration results from the quantitative method.

Fig. 23 shows the scaling curve on storage requirement indicated by Eq. (42). Assume that a posting occupies 4 bytes in space. For high-performance, we may decide to store the whole posting file in the random access memory (RAM). In recent years, a contemporary desktop computer may contain 1–4 GB RAM. We find that, for TREC/Blogs08 document collection, the whole partitioned posting file can be stored in RAM with 5–30 workstations. The ratio-to-ideal on the aspect of storage cost, indicated by Eq. (43) is given in Fig. 24. With the proposed posting file partitioning algorithm, the required storage cost is no more than 4% more than the ideal storage cost.

Fig. 25 shows the throughput scaling curve indicated by Eq. (40) for TREC/Blogs08. We draw the curve for the range of 5 to 30 workstations to fit storage capacity per workstation. The ratio-to-ideal on throughput aspect, indicated by Eq. (41), is shown in Fig. 26. Almost optimal throughput is achieved within the selected range on cluster size.

The evaluation result shows that, for TREC/Blogs08 document collection, the proposed partitioning algorithm results in a cluster with almost optimal throughput and storage cost. The reason is as follows. A system manager tends to select amount of workstations *M* such that

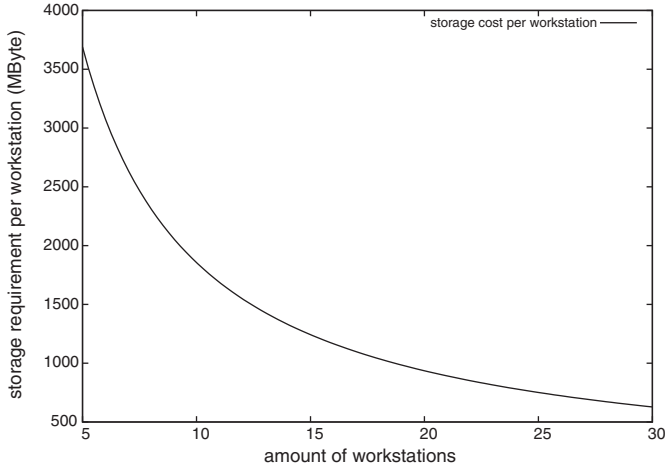


Fig. 23. Storage requirement scaling.

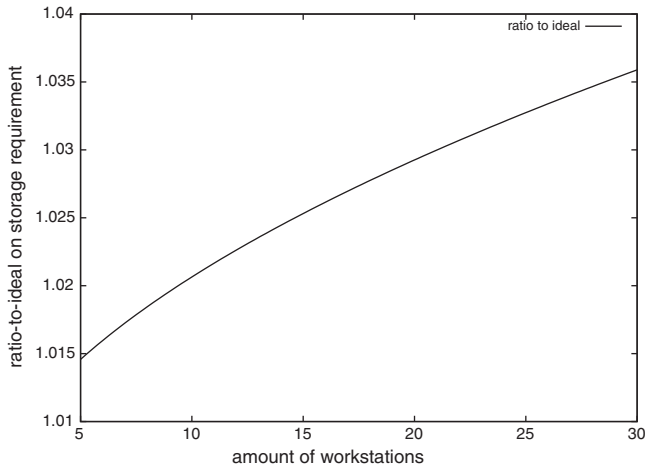


Fig. 24. Ratio-to-ideal scaling on storage requirement.

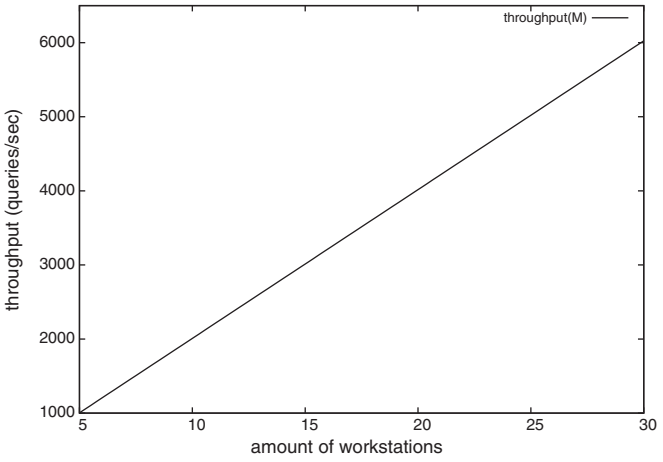


Fig. 25. Throughput scaling.

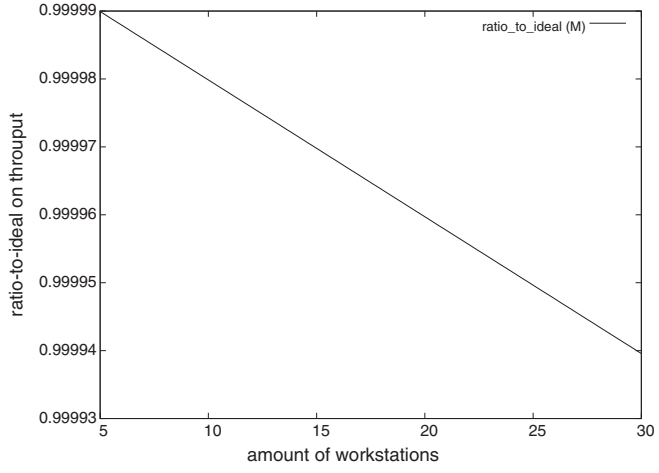


Fig. 26. Ratio-to-ideal on throughput aspect.

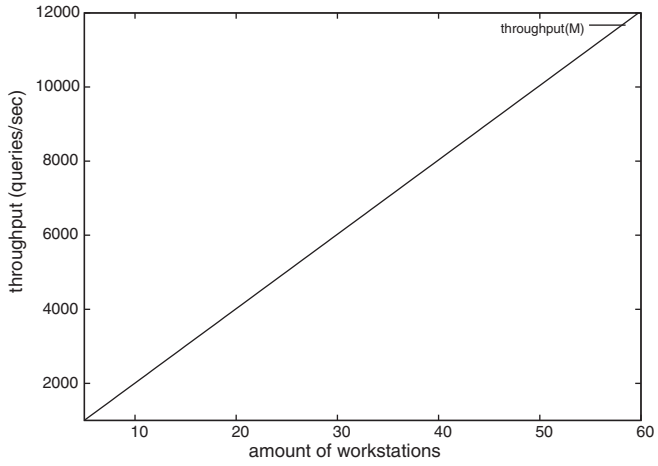


Fig. 27. Throughput scaling to meet ten thousands of requests per second.

(1) Storage capacity per workstation (*CAP*) approximates the balanced data size.

$$CAP \approx \frac{S}{M}.$$

(2) The required throughput λ is achieved when workload is uniformly distributed across all workstations.

$$\lambda \approx \frac{1}{TPP \times (L/M)}.$$

Analysis in Section 4 shows that the proposed algorithm approximates optimal load and storage balancing when

(1) the balanced data size is far larger than the size of a data item

$$\frac{S}{M} \gg \text{max. item size};$$

(2) the balanced load is far larger than the workload contributed by a data item

$$\frac{L}{M} \gg \max_{I_i}\{Load(I_i)\}.$$

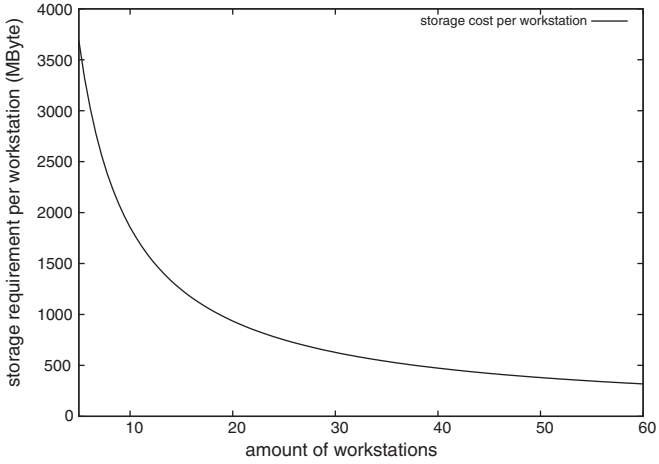


Fig. 28. Storage scaling for enlarged cluster size.

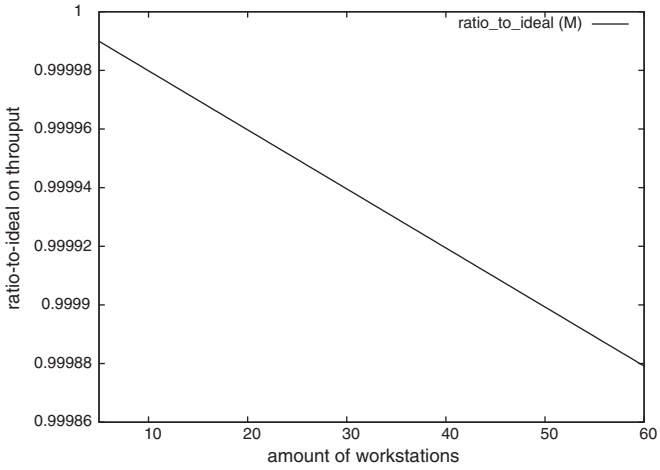


Fig. 29. Ratio-to-ideal for large throughput requirements.

Statistics data on TREC/Blogs08 shows that the above two conditions are met and we believe that this will be usual case for a large-scale document collection.

We also observe the performance for a cluster with higher throughput demand. Fig. 27 shows the throughput scaling curve to provide more than ten thousand queries per second. The cluster size is scaled up to contain 60 workstations. The scaling curve on storage requirement per workstation for the enlarged cluster is shown in Fig. 28. We find that, for such an enlarged cluster size, approximation factor on storage balancing is no longer a key design concern. The required storage space per workstation is within the RAM size that a contemporary computer will have. Moreover, our algorithm still approximates optimal throughput for the enlarged cluster size. Fig. 29 shows the ratio-to-ideal on throughput aspect. This result shows that our algorithm is still suitable for a clustered information retrieval system with high throughput demand.

9. Conclusions

This paper establishes the asymptotic 1-optimal result for static posting file partitioning. The partitioning considers all aspects on load balancing, storage balancing, and communication overhead. Communication overhead is avoided by partition-by-document-ID principle. The key result is that the algorithm is proved to be asymptotic 1-optimal on both load and storage balancing. The result indicates that, for large document collection, the algorithm achieves almost optimal result. Usefulness of the partitioning algorithm on real-world application is evaluated with TREC document collection. The partitioning algorithm is static in the sense that it partitions the whole inverted file at once and has to work off-line. For a Web search engine, the document collection growth rapidly and term popularities may change day by day. Based on the results in this paper, the future work is to adopt the theory to cope with dynamic changes on document collection and query log behavior.

The major impact of the results is that the effort to design a large-scale information retrieval system is simplified. In recent years, major Web search engines use large-scale clusters to store huge amount of data and handle high query arrival rate. Reducing storage cost as well as improving query processing throughput are required. Moreover, instead of running complex simulations, an analytical method to design a large-scale cluster is desired. In the previous work, storage efficiency was not considered and complex simulation was required for performance evaluation. Our algorithm has guaranteed load and storage balancing factor been proved and experiment shows that the guaranteed factors approximate optimal in real-world application. These results enable the system

manager to setup the cluster configuration from the proposed analytical model. With the success of this research, the traditionally ad-hoc approaches to the design of parallel information retrieval system can now become very systematic and analytical.

References

Arpaci-Dusseau, A., Arpaci-Dusseau, R., Culler, D.E., Hellerstein, J.M., Patterson, D., 1997. High performance sorting on networks of workstations. In: Proceedings of 1997 ACM SIGMOD Conference.

Arpaci-Dusseau, A., Arpaci-Dusseau, R., Culler, D.E., Hellerstein, J.M., Patterson, D., 1998. Searching for the sorting record: experiences in tuning now-sort. In: Proceedings of 1998 Symposium on Parallel and Distributed Tools.

Badue, C., Ribeiro-Neto, B., Baeza-Yates, R., Ziviani, N., 2001. Distributed query processing using partitioned inverted files. String Processing and Information Retrieval (November) 10–20.

Barroso, L., Dean, J., Holzle, U., 2003. Web search for a planet: the Google cluster architecture. Micro, IEEE 23 (March–April), 22–28.

Cacheda, F., Carneiro, V., Plachouras, V., Ounis, I., 2007. Performance analysis of distributed information retrieval architectures using an improved network simulation model. Information Processing and Management 43 (1), 204–224.

Cacheda, F., Plachouras, V., Ounis, I., 2005. A case study of distributed information retrieval architectures to index one terabyte of text. Information Processing and Management 41 (5), 1141–1161.

Dean, J., Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. Communications of the ACM 51 (1), 107–113.

Dowdy, W., Foster, D., 1982. Comparative models of the file assignment problem. ACM Computing Surveys 14 (2), 287–313.

Frakes, W.B., Baeza-Yates, R., 1992. Information Retrieval: Data Structures and Algorithms. Prentice-Hall.

Hardman, D.K., 1992. Proceedings of TREC Text Retrieval Conference.

Horowitz, E., Sahni, S., Rajasekaran, B., 1996. Computer Algorithms/C++. W.H. Freeman and Company.

Jeong, B.S., Omiecinski, E., 1995. Inverted file partitioning schemes in multiple disk systems. IEEE Transactions on Parallel and Distributed Systems 6 (2), 142–153.

Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R., Graham, R.L., 1974. Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM Journal on Computing 3 (4), 299–325.

Kumar, V.P., 1994. Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjamin Cummings Ltd.

Lee, H., Park, T., 1995. Allocating data and workload among multiple servers in a local area network. Information Systems 20 (3).

Lee, L.W., Scheuermann, P., Vingralek, R., 2000. File assignment in parallel i/o systems with minimal variance of service time. IEEE Transactions on Computers 49 (2), 127–140.

Little, T.D.C., Venkatesh, D., 1995. Popularity-based assignment of movies to storage devices and video-on-demand system. ACM/Springer Multimedia System 2 (6), 280–287.

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 ,. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, Belgium, pp. 1–9.

Macdonald, C., Santos, R.L., Ounis, I., Soboroff, I., 2010. Blog track research at TREC. SIGIR Forum 44 (1), 58–75.

MacFarlane, A., McCann, J.A., Robertson, S.E., 2000. Parallel search using partitioned inverted files. In: Proceedings of the 7th International Symposium on String Processing and Information Retrieval , pp. 209–220.

Moffat, A., Webber, W., Zobel, J., 2006. Load balancing for term-distributed parallel retrieval. In: SIGIR '06: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval ,. ACM Press, New York, NY, USA, pp. 348–355.

Moffat, A., Webber, W., Zobel, J., Baeza-Yates, R., 2007. A pipelined architecture for distributed text query evaluation. Information Retrieval 10 (3), 205–231.

Narendran, B., Rangarajan, S., Yajnik, S., 1997. Data distribution algorithm for load balanced fault-tolerant web access. In: Proceedings of 16th Symposium on Reliable Distributed Systems , pp. 97–106.

Riberio-Neto, B.A., Kitajima, J.P., Navarro, G., 1998. Parallel generation of inverted files for distributed text collections. In: Proceedings of the 18th International Conference of the Chilean Society of Computer Science , pp. 149–157.

Rotem, D., Schloss, G., Segev, A., 1993. Data allocation of multi-disk databases. IEEE Transactions on Knowledge and Data Engineering 5 (5), 877–882.

Salton, G., 1989. Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley.

Serpanos, D.N., Georgiadis, L., Bouloutas, T., 1998. MMPacking: a load and storage balancing algorithm for distributed multimedia servers. IEEE Transactions on Circuits and Systems for Video Technology 8 (1), 13–17.

Tomasic, A., Molina, H.G., 1995. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. IEEE Transactions on Parallel and Distributed Systems 6 (2), 142–153.

Wah, B., 1984. File placement on distributed computer systems. Computer 17 (1), 23–32.

Witten, I.H., Moffat, A., Bell, T.C., 1999. Managing Gigabytes: Compressing and Indexing on Documents and Images. Morgan Kaufmann Publishers, Inc.

Wolf, J., Pattipati, K., 1990. A file assignment problem model for extended local area network environments. In: Proceedings of 10th International Conference on Distributed Computing Systems, vol. 17, no. 1.

Zipf, G., 1949. Human Behavior and the Principle of Least Effort. Addison-Wesley.

Yung-Cheng Ma received the B.S and Ph.D. degree in computer science and information engineering from National Chiao-Tung University (NCTU), Hsinchu, Taiwan, in 1994 and 2002, respectively. After graduating from NCTU, he joined computers and communication laboratory of Industry Technology Research Institute (ITRI) in Taiwan as a hardware engineer. In ITRI, he leaded a group to develope embedded VLIW DSP processors. He joined Chang-Gung University at 2008 and is currently an assitant professor at Department of Computer Science and Information Engineering. His research interests include computer architecture, parallel and distributed systems, low-power embedded systems, and multi-core SoC design.

Tien-Fu Chen received the B.S. degree in Computer Science from National Taiwan University in 1983. After completed his military services, he joined Wang Computer Ltd., Taiwan as a software engineer for three years. From 1988 to 1993 he attended the University of Washington, receiving the M.S. degree and Ph.D. degrees in Computer Science and Engineering in 1991 and 1993 respectively. He is currently a Professor in the Department of Computer Science and Information Engineering at the National Chung Cheng University, Chiayi, Taiwan. In recently years, he has published several widely-cited papers on dynamic hardware prefetching algorithms and designs. His current research interests are computer architectures, distributed operating systems, parallel and distributed systems, and performance evaluation.

Chung-Ping Chung received the B.E. degree from the National Cheng-Kung University, Hsinchu, Taiwan, Republic of China in 1976, and the M.E. and Ph.D. degrees from the Texas A&M University in 1981 and 1986, respectively, all in electrical engineering. He was a lecturer in electrical engineering at the Texas A&M University while working towards the Ph.D. degree. Since 1986 he has been with the Department of Computer Science and Information Engineering at the National Chiao-Tung University, Hsinchu, Taiwan, Republic of China, where he is a professor. From 1991 to 1992, he was a visiting associate professor of computer science at the Michigan State University. From 1998, he joint the Computer and Communications Laboratories, Industrial Technology Research Institute, R.O.C. as the Director of the Advanced Technology Center, and then the Consultant to the General Director. He returned to his teaching position after this three-year assignment. His research interests include computer architecture, parallel processing, and parallelizing compiler.