行政院國家科學委員會專題研究計畫　成果報告

適用於 WEB 伺服器之可延伸式計算機系統設計(3/3)

計畫類別：個別型計畫
計畫編號：NSC91-2213-E-009-074-
執行期間：91 年 08 月 01 日至 92 年 07 月 31 日
執行單位：國立交通大學資訊工程學系

計畫主持人：鍾崇斌
共同主持人：單智君
計畫參與人員：馬詠程、謝萬雲、鄭哲聖

報告類型：完整報告

處理方式：本計畫可公開查詢

中　華　民　國 92 年 10 月 29 日

行政院國家科學委員會補助專題研究計畫　☑ 成果報告
　　　　　　　　　　　　　　　　　　□ 期中進度報告

適用於 WEB 伺服器之可延伸式計算機系統設計(3/3)

計畫類別：☑ 個別型計畫　　□ 整合型計畫

計畫編號：NSC－91－2213－E－009－074

執行期間：91 年 08 月 01 日至 92 年 07 月 31 日

計畫主持人：鍾崇斌

共同主持人：單智君

計畫參與人員：　馬詠程、謝萬雲、鄭哲聖

成果報告類型(依經費核定清單規定繳交)：□精簡報告　☑完整報告

本成果報告包括以下應繳交之附件：

□赴國外出差或研習心得報告一份

□赴大陸地區出差或研習心得報告一份

□出席國際學術會議心得報告及發表之論文各一份

□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢

□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：國立交通大學資訊工程學系

中華民國九十二年十月三十一日

中文摘要

最近幾年，許多熱門網站使用大型叢集系統(cluster)來儲存大量的資料並滿足效能的要求。這些叢集系統的主要應用為資訊檢索系統(information retrieval)，常見的有：(1)搜尋引擎(search engine)，(2)電子圖書館(Electronics Library)，(3)電子商務(E-Commerce)，(4)電子新聞(E-News)，以及(5)序列分析(Sequence Analysis)等等。網際網路的快速發展，為計算機系統的設計，帶來了全新的挑戰：資料配置須對執行效能與儲存空間作整體的考量，且須有計量方法以設計大型叢集資訊檢索系統。

在這篇論文中，我們提出一個兼顧負載與儲存量平衡的資料配置演算法。這演算法將大小不同的許多資料項目分配到工作站上。其理論基礎是 MMPacking 演算法。MMPacking 可以分配與複製相同大小的資料項目，以追求負載與儲存量的平衡。我們的主要構想是利用 bin packing 將大小不同的資料項的資料配置問題轉換、化簡成 MMPacking 所處理的問題。我們可以證明所提出的資料配置法是 asymptotical 1-optimal。當資料項目規模夠大時，所得的結果將趨近於最佳解。

利用所提的資料配置演算法，我們提出計量方法，以設計合乎成本效益的叢集式 Web 伺服器(clustered web server)。此計量方法的目的，是要降低滿足效能需求所需的硬體成本。硬體成本是以工作站的數量，及每部工作站的儲存空間大小來決定。為了達到此目的，須兼顧負載與儲存量的平衡。我們建立效能模型，以計算 throughput 與 response time。此計量方法大幅簡化設計大型叢集式資訊檢索系統的方法。

關鍵字：叢集式 Web 伺服器、搜尋引擎、轉置檔案、大型叢集式資訊檢索系統、資料配置演算法

英文摘要

In recent years, many major search engines on Internet use a large-scale cluster to store a large database and cope with high query arrival rate. A quantitative method to reduce the hardware cost is desired. This paper investigates 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. Hardware cost of the cluster depends on the cluster configuration: the number of workstations and storage capacity per workstation. The primary work to achieve the research objective is posting file partitioning. The objective of posting file partitioning is to minimize the storage requirement per workstation subject to a limited difference to ideal mean query processing time. The partitioning follows the partition-by-document-ID principle such that a query can be processed in parallel without transferring postings between workstations. Mean query processing time is estimated according to popularities of keyword terms. The kernel of the posting file partitioning algorithm is a data allocation algorithm, which maps variable-size items onto workstations for load and storage balancing. The data allocation algorithm is an integration of bin packing, MMPacking, and bin splitting procedures. We prove that the data allocation algorithm satisfies a load balancing constraint with asymptotically 1-optimal storage cost. With the posting file partitioning algorithm, we show a systematic approach to determine the cluster configuration. Evaluation with TREC document collection shows the usefulness of the proposed posting file partitioning algorithm on real-world applications. This research simplifies the effort to design a large-scale information retrieval system.

Keywords: clustered web server, search engine, posting file, large-scale IRS, data allocation algorithm

目錄

1 Introduction

This paper investigates 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, 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 fully considered the requirements of contemporary major search engines. Previous researchers [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12] investigate data allocation for high performance information retrieval. In these work, storage efficiency is not considered and 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 difference to ideal 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 is associated with a load and a data size, onto a set of workstations. The 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 dealing with variable-size items,

(2) a posting file partitioning model such that a query is processed in parallel without transferring postings between workstations, and

(3) a quantitative method to design a clustered search engine without complex simulations.

Usefulness of the posting file partitioning on real-world application is evaluated with TREC [13] 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, a conclusion is given in Section 8.

2 Background on Information Retrieval

The section describes concepts of information retrieval and analyzes the complexity to address research issues. An information retrieval system receives users queries and responses 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 unmatches a query in binary fashion. For each query, set operations ($\cap$, $\cup$, etc.) is 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 of 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

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

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

The answer list of a query is computed using the inverted file [14], [15]. An example of the inverted file is depicted in Figure 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 the 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 perform set operations on the posting lists of queried terms. For sorted list, the set operation result can be obtained by merging posting lists according to Boolean operators [16].

Time complexity of query processing is as follows. Time to search for the index file is no more than $O(m * \log n)$ [14] where m is the number of queried terms and n is the number of all indexed terms. Zipf's law [17] [16] states that 95% of words in documents fall in a vocabulary with no more than 8000 distinct terms. A query is typed by an user and hence m is usually small. Complexity on index file side is not critical. Let f_{t_i} be the length of the posting list of the queried term t_i. The time to retrieve and merge the posting lists is $O(f_{t_1} + f_{t_2} + ... + f_{t_m})$. 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.

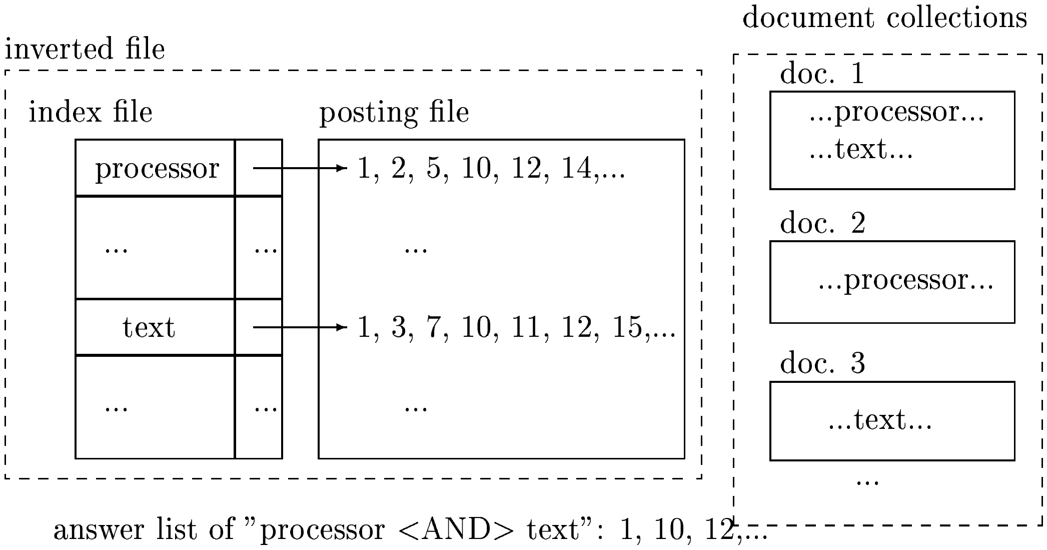


Figure 1: Inverted file

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 $I = \{I_0, I_1, ..., I_{N-1}\}$ and a set of workstations $WS = \{WS_0, WS_1, ..., WS_{M-1}\}$. Each item I_i is associated with a load, denoted $Load(I_i)$, and a size s_i. We normalize the size such that $0 < s_i \leq 1.00$ for each I_i and the size of the largest item is 1.00. 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 a limited difference to ideal load balancing. We formally specify an allocation to formulate the optimization problem.

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

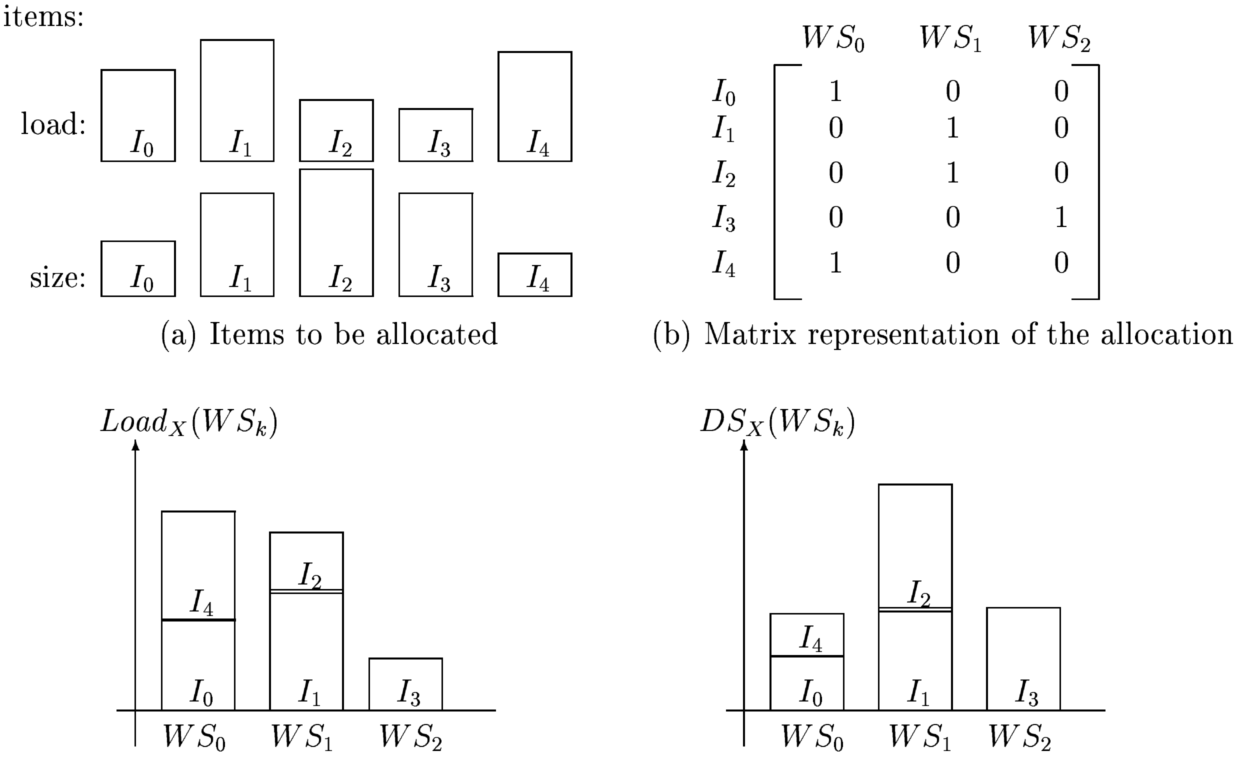
- each entry is either 0 or 1, and
- there exists an unique 1 in each row of X.

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

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

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

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



$$\begin{array}{c|ccc|} & WS_0 & WS_1 & WS_2 \\ I_0 & 1 & 0 & 0 \\ I_1 & 0 & 1 & 0 \\ I_2 & 0 & 1 & 0 \\ I_3 & 0 & 0 & 1 \\ I_4 & 1 & 0 & 0 \end{array}$$

(a) Items to be allocated

(b) Matrix representation of the allocation

(c) Load allocated on workstations

(d) Data size allocated on workstations

Figure 2: Example of an allocation without item replication

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

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

The constraint is that the difference to ideal load balancing is within the load of some item. That is,

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

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. Our survey is as follows. In early 1970's, researchers investigated data allocation for minimizing the storage cost [18]. From 1980's, needs in high performance database systems has turned the research focus to improve the data retrieval performance [19], [20], [21], [22], [23], [24], [25], [26]. Starting in late 1990's, 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 [27] proposed the MMPacking algorithm, which distributes and replicates identical-size data items onto workstations for both load and storage balancing. MMPacking [27] 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) optimizing for both load balancing and reducing storage cost,

(2) capable to deal with variable-size items, and

(3) not allowing item replication.

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

We propose a data allocation algorithm satisfies all these requirements. The key idea of the proposed algorithm is problem reduction to MMPacking with bin packing. We introduce MMPacking [27] and bin packing [28] in detail.

3.2.1 MMPacking

MMPacking [27] deals with the following optimization problem. The input is a set of n objects $B = \{B_0, B_1, ..., B_{n-1}\}$ with identical data sizes, and a set of M workstations $WS = \{WS_0, WS_1, ..., WS_{M-1}\}$. 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 $n * M$ matrix Y similar to the allocation matrix X defined in Section 3.1. The differences are that

- each entry in Y is a real number between 0 and 1, and

- there may be multiple non-zero entries in a row of Y.

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.

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

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 $\sum_{j=0}^{n-1} \lceil Y_{jk} \rceil$.

Figure 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 balanced load, 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 result matrix Y is shown in Figure 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 [27] have proved the following properties for MMPacking.

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

$$Load_Y(WS_k) = L/M$$

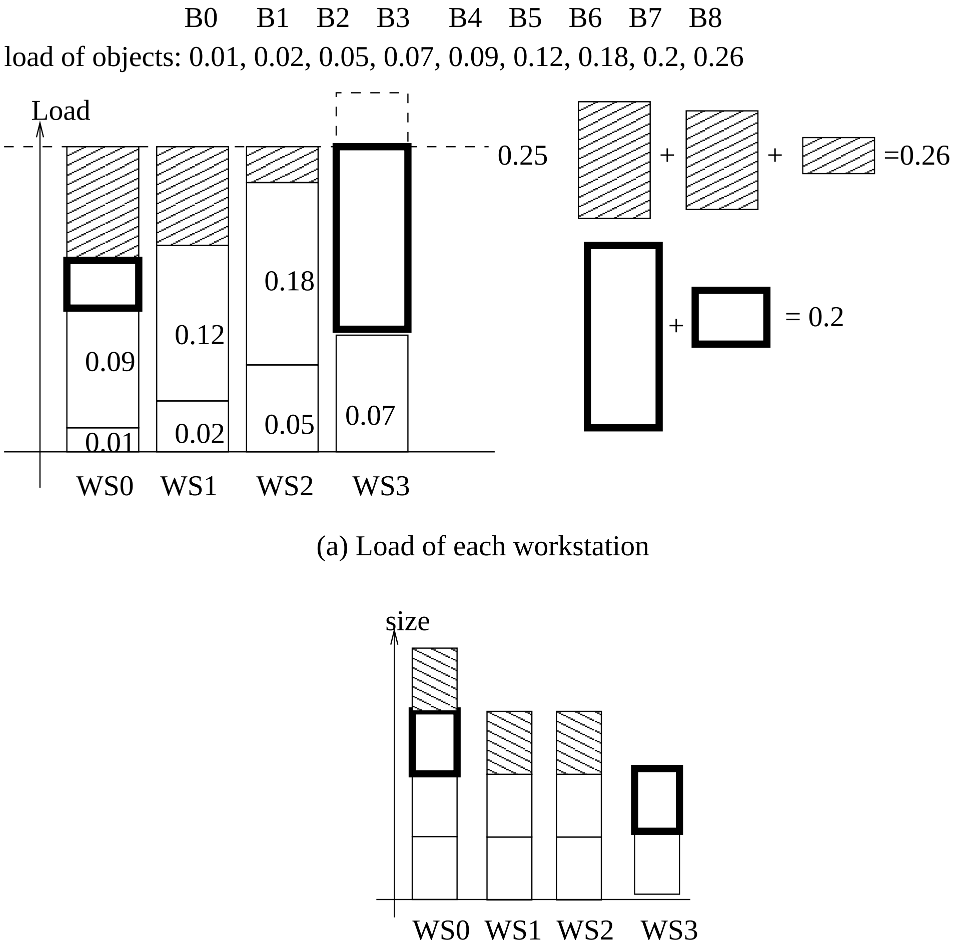
for any workstation WS_k.

Property 2 *The MMPacking algorithm allocates at least $\lfloor n/M \rfloor$ and at most $\lceil n/M \rceil + 1$ objects on an workstation.*

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.*

3.2.2 Bin packing

The bin packing problem [28] is as follows. The input is a set of items $I = \{I_0, I_1, ..., I_{N-1}\}$ 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 $B = \{B_0, B_1, ..., B_{n-1}\}$ with


B0 B1 B2 B3 B4 B5 B6 B7 B8
load of objects: 0.01, 0.02, 0.05, 0.07, 0.09, 0.12, 0.18, 0.2, 0.26
Load
0.25
0.09
0.01
0.12
0.02
0.18
0.05
0.07
WS0
WS1
WS2
WS3
+
+
=0.26
+
= 0.2
size
WS0
WS1
WS2
WS3

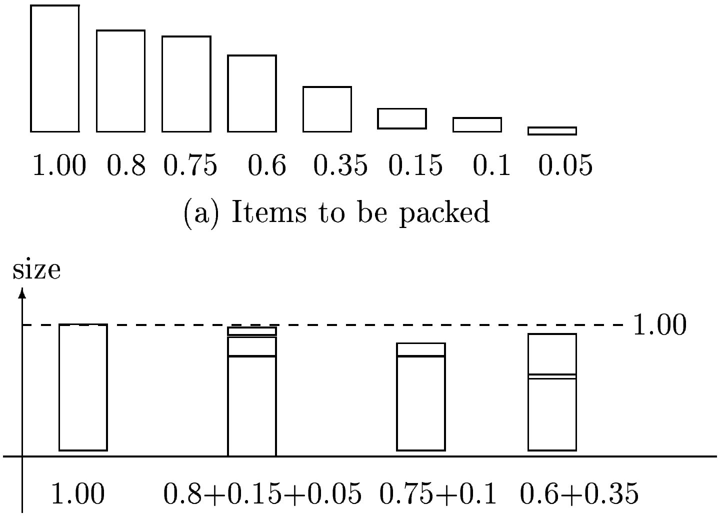
(a) Load of each workstation

(b) Data size stored in each workstation

Figure 3: Example of MMPacking

$$
\begin{array}{c|cccc|}
 & \text{WS0} & \text{WS1} & \text{WS2} & \text{WS3} \\
\text{B0} & 1.00 & 0.00 & 0.00 & 0.00 \\
\text{B1} & 0.00 & 1.00 & 0.00 & 0.00 \\
\text{B2} & 0.00 & 0.00 & 1.00 & 0.00 \\
\text{B3} & 0.00 & 0.00 & 0.00 & 1.00 \\
\text{B4} & 1.00 & 0.00 & 0.00 & 0.00 \\
\text{B5} & 0.00 & 1.00 & 0.00 & 0.00 \\
\text{B6} & 0.00 & 0.00 & 1.00 & 0.00 \\
\text{B7} & 0.10 & 0.00 & 0.00 & 0.90 \\
\text{B8} & 0.50 & 0.42 & 0.08 & 0.00
\end{array}
$$

Figure 4: Result Matrix of MMPacking



(a) Items to be packed

(b) Packing items with capacity=1.00

Figure 5: Example of bin packing

capacity x. Figure 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 [28] to perform bin packing. This algorithm iteratively places an item to a bin with the smallest room left. Property of the algorithm for analyzing our proposed algorithm is as follows.

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.*

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.

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

Phase 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:

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

Phase 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. Phase 1 packs variable-size items into bins with approximately equal size. Phase 2 performs MMPacking to determine the ideal load allocation and (approximately) balance amount of bins allocated on workstations. Phase 3 generates the final result X in which each item is allocated to an unique workstation. Figure 6 depicts the final result X generated from the MMPacking result Y shown in Figure 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 Phase 3 approximates the load sharing determined by MMPacking:

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

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 a complete algorithm.

4.1 Bin splitting for load balancing

This sub-section designs the bin splitting procedure and proves that the load balancing constraint (Eq.(4)) is satisfied.

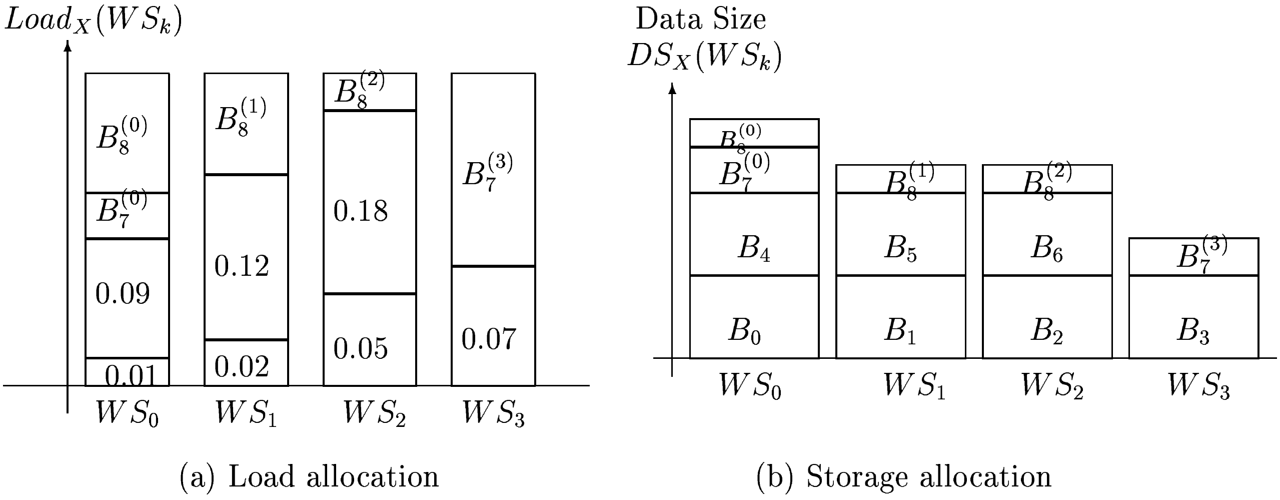


Figure 6: Example of final result generated from MMPacking result

4.1.1 Design of bin splitting procedure

The bin splitting procedure, named *SplitBin*, is in Figure 7. The objective of bin splitting is to approximate the load sharing determined by MMPacking (Eq.(7)). The procedure generates a partitioning PB_j of bin B_j according to the MMPacking result Y. The procedure examines each item $I_i \in B_j$ and makes slicing to generate $B_j^{(k)}$s in the order that MMPacking replicates B_j. A slicing is made whenever $Load(B_j^{(k)}) \geq Y_{jk} * Load(B_j)$. An example is shown in Figure 8, which depicts how the bin B_7 (with load 0.2) is split to approximate the MMPacking result shown in Figure 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. The key idea is to compare the load allocation between the final result X and the MMPacking result Y. The load of a workstation is rewritten for the comparison (Corollary 1). Let WS_k be a workstation sharing partial load of the bin B_j in the result of MMPacking. Load of $B_j^{(k)}$ is compared to $Y_{jk} * Load(B_j)$, the load sharing determined by MMPacking (Corollary 2 and 3). The load balancing property can then be derived (Theorem 1).

By observing Figure 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.*

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

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 makes a

Algorithm $\mathit{SplitBin}(B_j, Y, PB_j)$

- 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) $B_j^{(k)} \leftarrow \phi$ and $\mathit{Load}(B_j^{(k)}) \leftarrow 0$
    (3.2) while $\mathit{Load}(B_j^{(k)}) < Y_{jk} * \mathit{Load}(B_j)$ 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) $\mathit{Load}(B_j^{(k)}) \leftarrow \mathit{Load}(B_j^{(k)}) + \mathit{Load}(I_i)$
    (3.3) $k \leftarrow (k+1)$ mod $M$
```

Figure 7: Bin splitting procedure

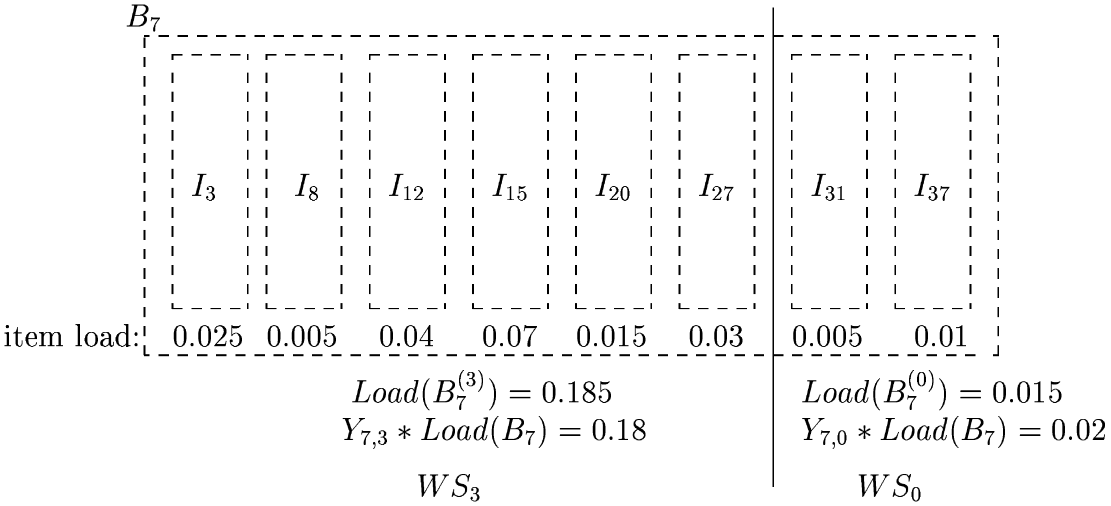

B_7
I_3
I_8
I_{12}
I_{15}
I_{20}
I_{27}
I_{31}
I_{37}
item load: 0.025 0.005 0.04 0.07 0.015 0.03 0.005 0.01
$\mathit{Load}(B_7^{(3)}) = 0.185$
$Y_{7,3} * \mathit{Load}(B_7) = 0.18$
WS_3
$\mathit{Load}(B_7^{(0)}) = 0.015$
$Y_{7,0} * \mathit{Load}(B_7) = 0.02$
WS_0

Figure 8: Example of bin splitting

slicing at the first place that the accumulated load is greater than or equal to the load sharing determined by MMPacking (cf. Step (3.2) in Figure 7). The difference between $Load(B_j^{(k)})$ and $Y_{jk} * Load(B_j)$ is thus at most the load of some item (stated by 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) $Y_{jk} * Load(B_j)$. Total load of all portions is fixed:

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

Hence (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 sharing 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.*

$$Load(B_j^{(k)}) \leq Y_{jk} * Load(B_j) + \max_{I_i}\{Load(I_i)\} \tag{9}$$

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.*

$$Load(B_j^{(k)}) \leq Y_{jk} * Load(B_j) \tag{10}$$

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).

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

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

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 that a workstation contains two replicated bins. The remaining cases are similar and omitted.

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

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

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 Corollary 2 and 3, we have:

$$Load(B_{j_1}^{(k)}) \leq Y_{jk} * Load(B_{j_1}) + \max_{I_i}\{Load(I_i)\}$$

$$Load(B_{j_2}^{(k)}) \leq Y_{jk} * Load(B_{j_2})$$

Load allocated on WS_k satisfy the following equation.

$$Load_X(WS_k) \leq \sum_{B_j:Y_{jk}>0} Y_{jk} * Load(B_j) + \max_{I_i}\{Load(I_i)\}$$

MMPacking achieves exact load balancing (Property 1).

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

Eq.(11) is thus obtained. **Q.E.D.**

4.2 Bin capacity selection for storage balancing

We derive equations to select the bin capacity and indicate an upper bound on the allocated data size for any workstation. Two cases are analyzed: (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 elementary bin capacity

We first consider the case of setting bin capacity $x = 1$, the size of the largest item, for bin packing. Bin packing generates a set of bins with sizes exceeding 1/2 except for the smallest bin (Lemma 1). The number of bins generated are bounded (Lemma 2), and the upper bound on allocated data size is obtained (Theorem 2).

Lemma 1 *With bin capacity $x = 1$, there is at most one bin filled with size less than 1/2 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. The basis, after packing item I_0, is trivial. Suppose the lemma holds after packing I_i. The lemma holds again after the packing of I_{i+1} if no new bin is initialized for I_{i+1}. Consider the case that a new bin is initialized to pack I_{i+1}. If size of each bin is at least 1/2 after the packing of I_i (see Figure 9(a)), the new bin is the only possible one with size not exceeding 1/2 after the packing of I_{i+1}. In case that there is a unique bin B_j with size less than 1/2 after the packing of I_i (see Figure 9(b)), according to Property 4, each of the initialized bin has no room for I_{i+1} and hence size $s_{i+1} \geq 1/2$. The size of the new bin will exceed 1/2 and B_j is still the only one with size not exceeding 1/2 after the packing of I_{i+1}. The lemma holds again after the packing of I_{i+1} for all possible cases. **Q.E.D.**

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.

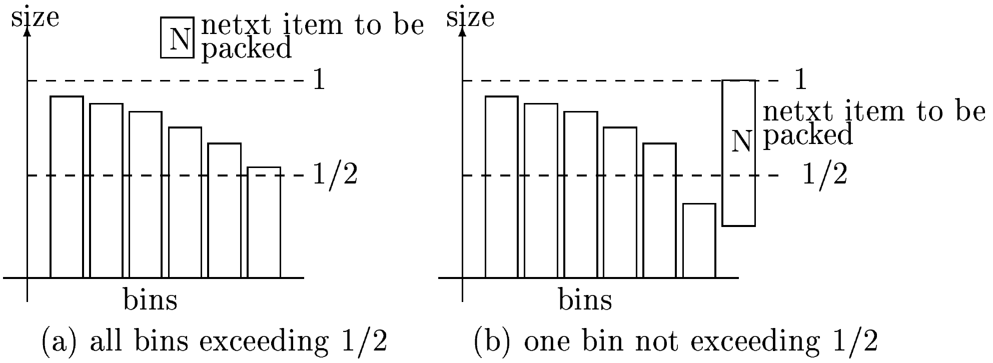


Figure 9: Packing an item into bins with capacity one

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

$$n \leq 2 * S + 1 \tag{13}$$

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.

$$S \geq \frac{1}{2} * (n - 1)$$

Equation (13) is thus obtained. **Q.E.D.**

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

Theorem 2 (Storage requirement of setting elementary bin capacity) *With bin capacity $x = 1$, the proposed algorithm generates an allocation X in which*

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

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 data size of a bin is at most 1.00. Hence we have the upper bound on the data size allocated on WS_k:

$$DS_X(WS_k) \leq \lceil n/M \rceil + 1.$$

Total number of bins n is bounded from above as stated in Eq.(13). Eq.(14) is thus obtained. **Q.E.D.**

4.2.2 Case of enlarged bin capacity

We improve storage balancing by enlarging bin capacity. Figure 10 depicts an example of no items being packed together if bin capacity is set to be 1. (Recall that size is normalized such that the size of the largest item is 1.00 and size $s_i \leq 1$ for any item I_i.) In this example, some items are with size 1.00 and some are with size slightly greater than 1/2. In the worst case, the allocated data size of the workstation containing most data approaches twice the data size allocated on the workstation containing least data. However, the worst case can easily 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.

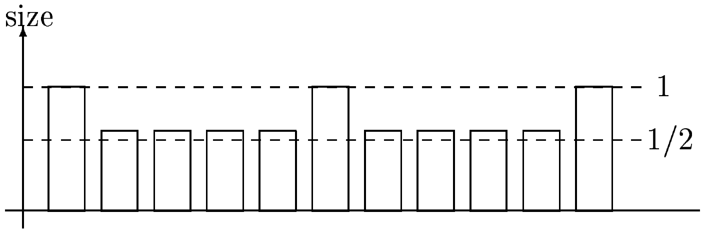


Figure 10: Worst case of setting elementary bin capacity

Selecting bin capacity encounters a tradeoff. Suppose the bin capacity is set to be $x > 1.00$. Let n be the number of bins generated and M be the number of workstations. Figure 11 depicts maximum difference on allocated data size between workstations. MMPacking [27] allocates $\lfloor n/M \rfloor$ to $\lfloor n/M \rfloor + 2$ bins on each workstation. Except for the smallest bin, the size of a bin lies between $x-1$ and x (Lemma 3). Data size allocated on a workstation is bounded from above as follows.

$$\max_{WS_k}\{DS_X(WS_k)\} \leq \left(\left\lfloor \frac{n}{M} \right\rfloor + 2\right) * x$$

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

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

Difference on allocated data size between workstations is as follows.

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

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

The analysis is outlined as follows. Lemma 3 states packed bin sizes. Lemma 4 bounds number of generated bins according to packed bin sizes. With the bound on

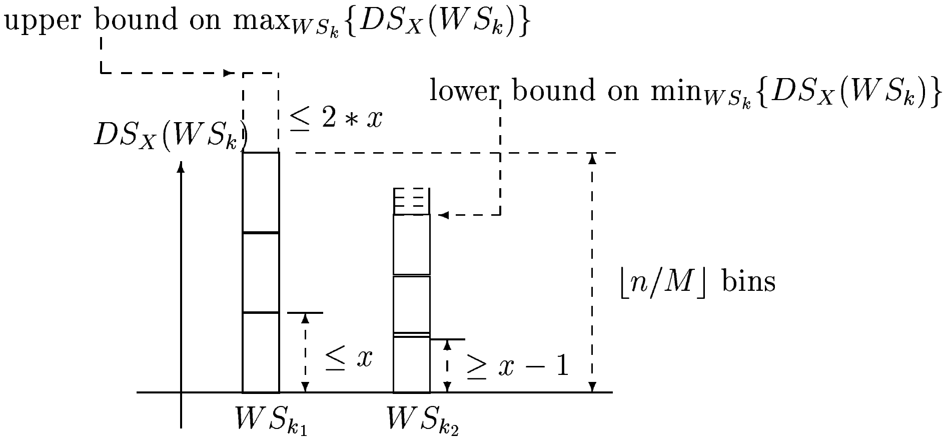

upper bound on $\max_{WS_k}\{DS_X(WS_k)\}$
lower bound on $\min_{WS_k}\{DS_X(WS_k)\}$
$DS_X(WS_k)$
$\leq 2*x$
$\lfloor n/M \rfloor$ bins
$\leq x$
$\geq x-1$
WS_{k_1}
WS_{k_2}

Figure 11: Effects of bin capacity selection on storage balancing

number of generated bins, Lemma 5 relates storage requirement to selected bin capacity and defines the *capacity function* (Eq.(18)). The bin capacity x is selected to minimize the capacity function. The storage requirement for a workstation can also be derived (Theorem 3).

Lemma 3 *With bin capacity $x > 1$, there is at most one bin filled with size less than $x - 1$ 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. The basis, status after packing item I_0, is trivial. Suppose the lemma holds after the packing of I_i. There are two cases for the status after packing I_i: (i) size of each bin exceeds $x-1$ (Figure 12(a)), and (ii) there is a unique bin B_j with size not exceeding $x-1$ (Figure 12(b)). For case (i), the new bin (if initialized) is the only possible one with size less than $x-1$ after packing I_{i+1}. For case (ii), no new bin will be initialized (Property 4) since size $s_{i+1} \leq 1$ and at least B_j has room for I_{i+1}. B_j is the only possible bin with size not exceeding $x-1$ after packing I_{i+1}. The lemma holds again after the packing of I_{i+1}. **Q.E.D.**

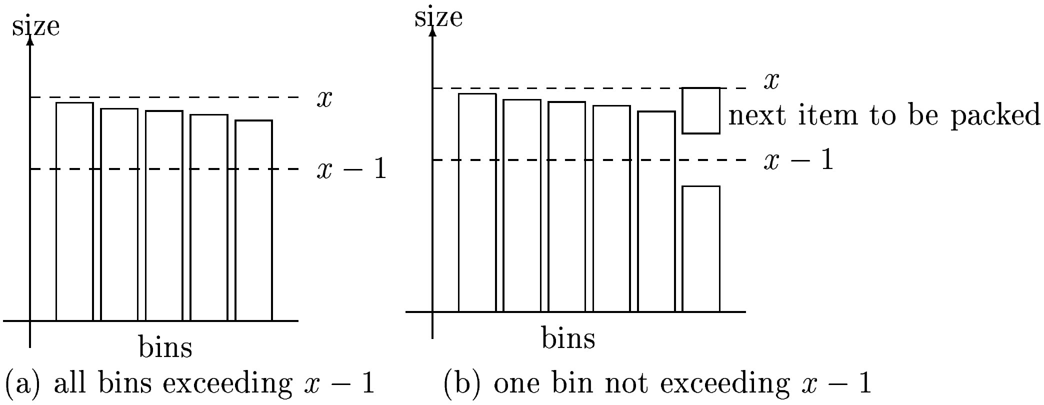
Similar to Lemma 2, we derive the upper bound on the number of bins generated. (S is the total size of all items.)

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

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

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,

$$S \geq (x-1)*(n-1)$$



(a) all bins exceeding $x-1$ (b) one bin not exceeding $x-1$

Figure 12: Packing an item to bins with enlarged bin capacity

Eq. (16) is obtained immediately. **Q.E.D.**

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*

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

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 sizes of all bins not exceeding x.

$$DS_X(WS_k) \leq (\lceil n/M \rceil + 1) * x$$

Lemma 4 gives an upper bound on n and Eq. (17) is obtained immediately. **Q.E.D.**

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

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

The curve of the capacity function on the x-y plane is depicted in Figure 13, which reflects the tradeoff on selecting the bin capacity. Taking differential on $f(x)$ and solving the equation $f'(x) = 0$, we obtain the **optimal bin capacity** x_0 to minimize $f(x)$:

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

And the required storage capacity for a workstation is obtained.

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

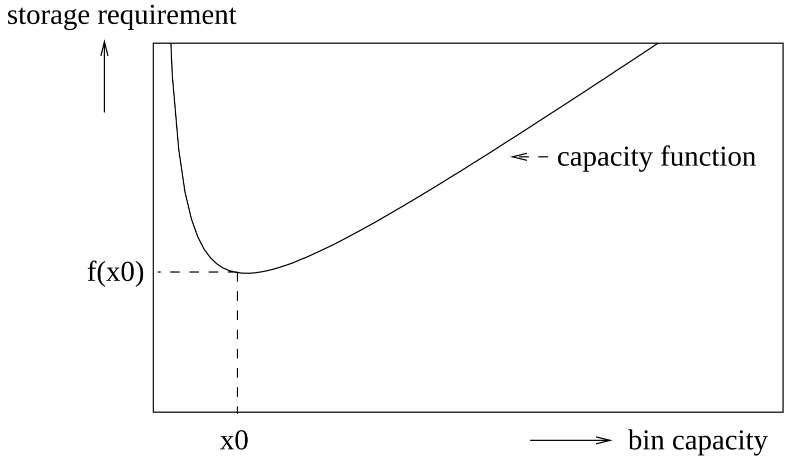


Figure 13: Capacity function for selecting bin capacity

Theorem 3 (Storage requirement of enlarged bin capacity) *By selecting bin capacity $x = 1 + \sqrt{\frac{S}{3*M}}$, the proposed algorithm generates an allocation X in which*

$$DS_X(WS_k) \leq \frac{S}{M} + 2\sqrt{3} * \sqrt{\frac{S}{M}} + 3 \qquad (21)$$

for any workstation WS_k.

Proof. This is the conclusion of previous discussion. **Q.E.D.**

The theorem indicates the optimization quality of the proposed algorithm. The ideal data size allocated on a workstation is S/M. Eq.(21) indicates the relation between the optimal solution and the solution found by the proposed algorithm.

4.3 Summary of Proposed Algorithm

We summarize previous discussion to form 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 in Figure 14. The algorithm determines the bin capacity according to Section 4.2. By comparing Eq.(14) and Eq.(21), whether to enlarge the bin capacity or not can be determined according to the problem parameter S/M, where S is the total data size and M is the number of workstations. (The two equations Eq.(14) and Eq.(21) equal at $S/M = 12$.) Properties of the output are proved in previous sections.

The complexity of the data allocation algorithm is as follows. Let N be the number of items and M be the number of workstations. The time complexity of best-fit bin packing is $O(N^2)$ [28]. The time complexity for the MMPacking to allocate n ($\leq N$) bins is $O(n + M)$ [27]. Hence the time complexity of the proposed algorithm is $O(N^2 + M + n)$. To implement the algorithm, an $O(n * M)$ space is required to store the result of MMPacking Y. The final result X can be implemented as a mapping table with $O(N)$ space. Hence the space complexity of the algorithm is $O(n * M + N)$.

```
Algorithm $LSB\_Alloc(I, WS, X)$
  • Input:
      – a set of data items $I = \{I_0, I_1, ..., I_{N-1}\}$, each item $I_i$ is associated with
        $Load(I_i)$ and data size $s_i$
      – a set of $M$ workstations $WS = \{WS_0, WS_1, ..., WS_{M-1}\}$
  • Output: an allocation $X$ of $I$ onto $WS$ with the following properties:
      – Load balancing: $Load_X(WS_k) \leq (L/M) + \max_{I_i}\{Load(I_i)\}$ for any $WS_k$
            (where $L$ is the total load of all items)
      – Storage balancing:
          * if $S/M \leq 12$, $DS_X(WS_k) \leq 2 * (S/M) + 3$ for any $WS_k$
          * if $S/M > 12$, $DS_X(WS_k) \leq (S/M) + 2\sqrt{3} * \sqrt{(S/M)} + 3$ for any $WS_k$
            (where $S$ is the total data size)
  • Method:
    (1) /* bin-packing phase */
        (1.1) $S \leftarrow \sum_{I_i} s_i$
        (1.2) if $S/M < 12$ then $x \leftarrow 1$ else $x \leftarrow 1 + \sqrt{S/(3 * M)}$
        (1.3) perform bin packing to pack $I$ into a set of bins $B$ with capacity $x$
    (2) /* MMPacking phase */
        (2.1) $Load(B_j) \leftarrow \sum_{I_i \in B_j} Load(I_i)$ for each bin $B_j \in B$
        (2.2) perform MMPacking to obtain allocation $Y$ which allocates $B$ on $WS$
    (3) for each $B_j \in B$ do
          – if $\exists WS_k$ such that $Y_{jk} = 1$ then setting $X$: allocating all $I_i \in B_j$ to
            $WS_k$
          – else
            (3.1) perform $SplitBin(B_j, Y, PB_j)$
            (3.2) for each $B_j^{(k)} \in PB_j$ do setting $X$: allocating all $I_i \in B_j^{(k)}$ to $WS_k$
```

Figure 14: Proposed data allocation algorithm

The algorithm is asymptotically 1-optimal on storage balancing. Let J be an instance (a pair of 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 to optimal is bounded as follows.

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

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

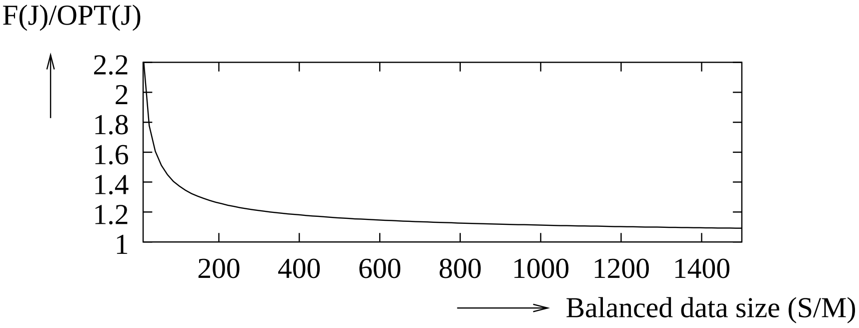


Figure 15: Storage balancing property of proposed algorithm

5 Parallel Information Retrieval

We turn back to information retrieval. Previous sections propose the data allocation algorithm. Remaining work is to apply the data allocation algorithm for posting file partitioning. An issue is to specify what an 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 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 the principle, a query is processed in parallel without transferring postings between workstations. Time complexity of parallel query processing is 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 explained in Figure 16. The principle is that all postings referring to the same document ID has to be allocated on the same workstation. Each workstation covers a set of document IDs. In parallel query processing, workstation WS_k is responsible for picking out answers from document IDs covered by it. For example, for the query (term 1 <AND> term 2), WS_1 has to pick out $\{4, 8\}$ from $\{1, 4, 7, 8\}$. Checking whether a document ID d matches a query or not requires only postings referring to document ID d. 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 in parallel without transferring postings between workstations.

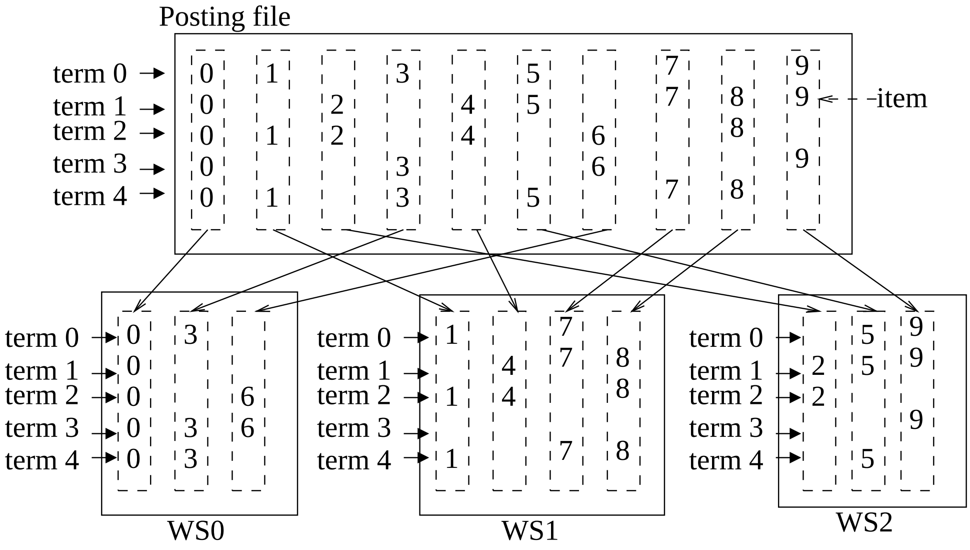


Figure 16: Partition-by-document-ID principle

Remaining part of this section describes how a query is processed in parallel. Section 5.1 deals with the set-theoretic foundation to derive the time complexity of parallel query processing. Section 5.2 deals with implementation issues on cluster of workstations.

5.1 Theory of parallel query processing

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 and 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 term t. For the example in Figure 16, local posting files for all workstations are depicted in Figure 17.

WS_0	WS_1	WS_2
term0 → 0,3	term0 → 1,7	term0 → 5,9
term1 → 0	term1 → 4,7,8	term1 → 2,5,9
term2 → 0,6	term2 → 1,4,8	term2 → 2
term3 → 0,3,6	term3 →	term3 → 9
term4 → 0,3	term4 → 1,7,8	term4 → 5

Figure 17: Example of local posting files

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 of 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 written as 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 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 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 written as 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 omitted. This theorem is proved by induction. **Q.E.D.**

Theorem 4 states an efficient way to compute a partial answer list. To compute $ANS_q(WS_k)$, WS_k has only to perform ordinary set operations on its local posting lists of queried terms. The list $ANS_q(WS_k)$ can be computed without examining all document IDs mapped to WS_k. An example is as follows. Consider the partitioned posting file in Figure 16. The given query q is "(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 [16] 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 [16], the time complexity to compute* $ANS_q(WS_k)$ *is* $O(f_{t_1}^{(k)} + f_{t_2}^{(k)} + ... + f_{t_m}^{(k)})$.

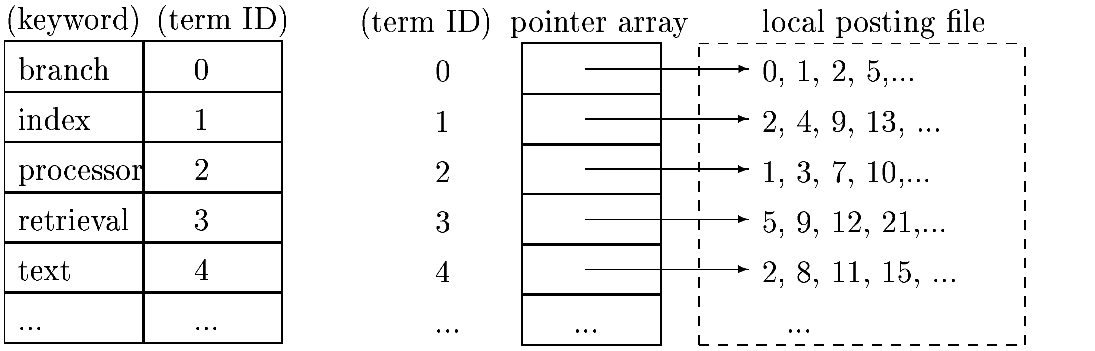
5.2 Implementation on cluster of workstations

This subsection describes the flow of processing a query on a network of workstations, starting from receiving a user query to the reply of the answer list. Boolean query processing is the major concern. With parallel sorting, the proposed scheme can also be extended for ranking.

The flow to compute the answer list of a query is as follows. A specific workstation, called the gateway, is responsible for receiving user queries and performing the index file search. The gateway searches the index file shown in Figure 18(a), and substitutes a term ID for each term in the query. Records of frequently used terms are often stored in the random access memory such that the average index search time will not scale with the size of the document 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 Figure 18(b). While receiving a broadcasted query, a workstation to retrieve local posting lists and computes its own partial answer list. The partial answer list is buffered locally, and number of answers found is sent back to the gateway.

The remaining work is to reply answers to the user page by page. A page contains the number of answers to the query, and a few titles of matched documents. The number of answers is useful for a user to determine whether to browse further matches. 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 in a page. Each



(keyword)	(term ID)
branch	0
index	1
processor	2
retrieval	3
text	4
...	...

(a) Index file in the gateway

(b) Local posting file in back-end workstation

Figure 18: Partitioned inverted file in cluster of workstations

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 sort [29] of all workstations' top r answers. With architectural support, Patterson's group shows that more than 1 Giga integers can be sorted in 2.41 seconds using 64 workstations [30], [31]. The time to rank answers for a page is small since r is small and will not scale with the collection size.

6 Posting File Partitioning Algorithm

We are now ready to complete the posting file partitioning algorithm. The input includes

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

The output is a partitioned posting file of PF_{seq} to be distributed on WS, following the partition-by-document-ID principle. Mean query processing time of parallel processing with 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 the ideal value plus the effect of an document. 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 is formulated as the data allocation problem defined in Section 3.1. Three rules are given to specify (1) what an item is, (2) size of an item, and (3) the 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 *The item I_i is the set of all postings referring to doc. ID i.*

With the rule, a partitioned posting file is generated as follows. The data allocation algorithm generates an allocation matrix X. For the allocation X, local posting list of term t at workstation WS_k, denoted $L_t(WS_k)$, is as follows.

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

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.

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

$$s_i = \frac{\textit{number of postings referring to doc. ID i}}{\max_{\textit{doc. ID j}}\{\textit{number of postings referring to doc. ID j}\}}. \tag{27}$$

Storage requirement of a workstation is calculated as follows. The size s_i indicates the number of postings in item I_i and is normalized such that $0 < s_i \leq 1.00$. Size of the largest item, the item containing most postings, is 1.00. 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

$$E[TQ_t] = 1 * Pr\{TQ_t = 1\} + 0 * Pr\{TQ_t = 0\} = p_t. \tag{28}$$

Let $f_t^{(k)}$ be the length of the local posting list of term t in workstation WS_k. Time quantity is normalized such that one unit of time is the average time to process a posting. 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.

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

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.

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

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

(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:*

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

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

Theorem 5 (Mean query processing time of parallel processing) *Mean query processing time of a workstation WS_k is the total 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_ts corresponding to postings. Consider Figure 16 for the imagination. $QPT_X(WS_k)$ can be calculated by scanning the local posting file row by row. Each time a posting is found, the corresponding TQ_t is accumulated. 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. **Q.E.D.**

Theorem 6 (Mean query processing time of sequential processing) *Mean query processing time of 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. **Q.E.D.**

With these three rules, Algorithm *LSB_Alloc* is applied to generate a partitioned posting file with the following properties. The three rules specify input to Algorithm *LSB_Alloc.* Algorithm *LSB_Alloc* 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 Theorem 2 and 3. Mean query processing time of parallel processing is stated in the following Corollary, which is a direct consequence of Theorem 1, 5, and 6.

Corollary 5 *Applying Algorithm LSB_Alloc generates a partitioned posting file such that*

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

for any workstation WS_k.

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

6.2 Generation of partitioned posting file

Figure 19 describes the algorithm to generate the partitioned posting file. The first phase scans the input posting file to set parameters of items. The proposed data allocation algorithm 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 in the algorithm *LSB_Alloc* and $O(f)$ is spent to scan the input posting file twice. The space complexity is $O(n * M + N + f)$: $O(f)$ space to store the input and generated posting file and $O(n * M + N)$ space for algorithm *LSB_Alloc.*

7 Application: Quantitative Method for Cluster Design

With the proposed posting file partitioning algorithm, we show a quantitative method to design a parallel information retrieval system systematically. The quantitative method

Algorithm $PostingFilePartition(PF_{seq}, TP, WS, PPF)$

- 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, consists of local posting list $L_t(WS_k)$ for each term t
- Method:

```
(1) /* scan $PF_{seq}$ to setup parameters for 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
              (3.2.1) let $WS_k$ be the workstation that $X_{ik} = 1$
              (3.2.2) append document ID $i$ to $L_t(WS_k)$
```

Figure 19: Algorithm for generating partitioned posting file

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 a workstation. We show the usefulness of our work on real-world applications with TREC document collection [13].

7.1 Cluster configuration problem

The concerned problem is to determine the 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 of sequential query processing, and
- throughput requirement λ.

The output, cluster configuration, specifies

- 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*(cost per workstation)]. The cost (price) per workstation depends on the storage capacity CAP. The objective is to minimize the hardware cost subject to the following constraints:

- throughput capability of the cluster $\geq \lambda$, and
- amount of data allocated on a workstation can be fit into the selected storage capacity CAP.
$$DS_X(WS_k) \leq CAP \text{ for any } WS_k$$

The following assumption is imposed on the given throughput requirement.

Assumption 1 *The given throughput requirement λ satisfies the following equation.*

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

Recall that the load of an item I_i is the quantity in mean query processing time imposed by document i. Note that time unit is normalized that one unit of time is average time per posting. The assumption ensures the existence of a solution to the cluster configuration problem (see the next subsection).

7.2 Calculation of cluster configuration

Cluster configuration is calculated according to load and storage balancing property 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

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

To cope with query arrival rate 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. When throughput requirement is given and M is to be determined, load balancing is to minimize M to achieve 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:

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

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

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

Note 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.

7.3 Usefulness on real world applications

We demonstrate the usefulness of our work on real-world applications with TREC [13] document collection. The evaluation metric is ratio to ideal,

$$\frac{\text{required number of workstations with proposed posting file partitioning algorithm}}{\text{number of workstations with ideal load/storage balancing}},$$

with throughput requirement and storage capacity per workstation been controlled parameters. The metric indicates the efficiency on using processing and storage capabilities of workstations. Query log on TREC document collection represents the behavior of real-world applications and determines input parameters for calculating the evaluation metric (Eq.(40) and (41)).

Parameters to calculate evaluation metrics are obtained as follows. The test data is a query log over TREC [13] document collection. Queries are generated randomly following [32]. Stop words (such as a, the, this, that, etc.) are eliminated from the documents. A query is formed by randomly selecting consecutive words from documents and inserting Boolean operators (AND,OR) into the consecutive words. A query contains 2 to 10 terms. A sequential information retrieval system is implemented on a PC with Linux operating system to obtain query processing time. A posting consumes 12 bytes: 4 bytes integer for document ID and 8 bytes floating point of weight for ranking. Table 1 shows the obtained parameters. The average time per posting includes disk access time. Note that these parameters are independent of the database scale. The experiment is designed to indicate the behavior of our work when applying to a database far larger than TREC [13].

average time per posting	0.606 us
(normalized) $\max_{I_i}\{Load(I_i)\}$	0.37
largest item size	24984 bytes (2082 postings)

Table 1: Parameters from TREC for calculating evaluation metrics

The evaluation metric for the efficiency on using throughput capabilities is as follows. Let M_{th-req} be the number of workstations for the proposed partitioning algorithm to satisfy the throughput requirement λ. The parameter M_{th-req} is determined by Eq.(38). Let $M_{th-ideal}$ be the number of workstations to satisfy throughput requirement λ with ideal load balancing.

$$M_{th-ideal} = \lambda * MQPT_{seq}$$

The evaluation metric is the ratio to ideal.

$$\frac{M_{th-req}}{M_{th-ideal}} = \frac{1}{1 - \lambda * \max_{I_i}\{Load(I_i)\}} \tag{40}$$

Note that parameter λ in Eq.(40) is normalized by the average time per posting.

Figure 20 depicts the evaluation results on the efficiency of using processing capabilities. Google [33] reports that the average query arrival rate is several thousands per second. The examined throughput requirement ranges up to ten thousand queries per second. Almost ideal efficiency is achieved since the effect of a document on mean query processing time is negligible. Corollary 5 states that almost ideal mean query processing time is achieved if the effect of a document is small. Our experiment shows that adding a document increases the mean query processing time by at most (0.606*0.37=0.224 us), which is quite small compared to the required mean query processing time even when 10000 queries arrived per second.

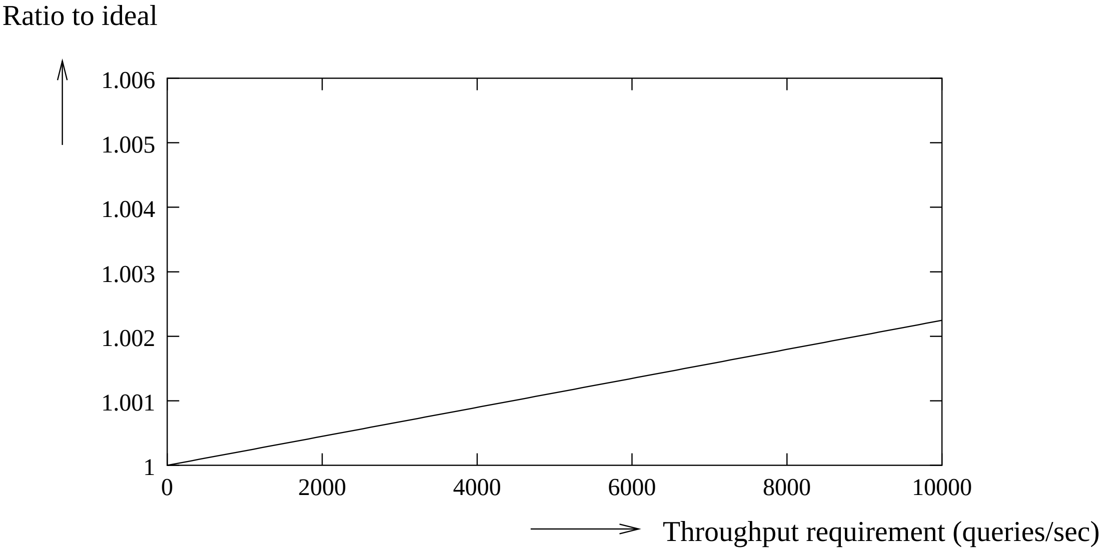


Figure 20: Efficiency on using processing capability

The evaluation metric for the efficiency on using storage capability is as follows. Let S be the total data size and CAP be the storage capacity per workstation. Let M_{st-req} be the required number of workstations to store all data with proposed posting file partitioning algorithm. The parameter M_{st-req} is determined by Theorem 3.

$$M_{st-req} = \frac{S}{(\sqrt{CAP} - \sqrt{3})^2}$$

Let $M_{st-ideal}$ be the number of workstations to store all data with ideal storage balancing.

$$M_{st-ideal} = \frac{S}{CAP}$$

The evaluation metric is the ratio to ideal.

$$\frac{M_{st-req}}{M_{st-ideal}} = \frac{CAP}{(\sqrt{CAP} - \sqrt{3})^2} \tag{41}$$

Note that CAP in above equations is normalized with the largest item size.

The evaluation results on the efficiency of using storage capacity are depicted in Figure 21 and 22. Both operating on random access memory (RAM) and operating on disks are evaluated. (Some search engines, such as Google [33], operate on RAM.) The RAM size of a contemporary PC ranges from 64MB to 2GB. The capacity of a contemporary disk ranges from 10GB to 100 GB and a PC can connect up to 7 disks. Almost ideal efficiency is achieved since the size of an item is far less than the storage capacity per workstation. An item is at most several Kilobytes but the storage capacity of a workstation is at least several Megabytes (even using only RAM). The storage requirement is determined by Theorem 3 and almost optimal storage cost is achieved.

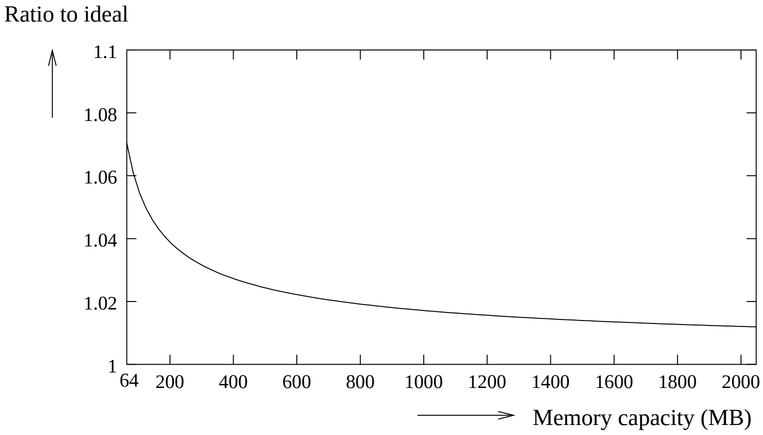


Figure 21: Efficiency on using storage capability (operating on RAM)

8 Conclusion

This paper investigates posting file partitioning for both high performance and storage efficiency. The research objective is to design a cluster satisfying throughput requirement with minimum hardware cost. Primary results are as follows. The kernel of the posting file partitioning algorithm is a data allocation algorithm for load and storage balancing. The proposed data allocation algorithm is an integration of bin packing [28], MMPacking [27], and bin splitting procedures. The algorithm is proved to satisfy the load balancing criteria with asymptotically 1-optimal storage cost. Based on the analysis, a quantitative method for the research objective is derived. Evaluation with TREC document collection [13] shows that, for real-world applications, the proposed algorithm achieves almost optimal efficiency on using processing and storage capabilities of workstations.

These results greatly simplifies the effort to design a large-scale information retrieval system. In recent years, many major search engines on Internet use a cluster of hundreds

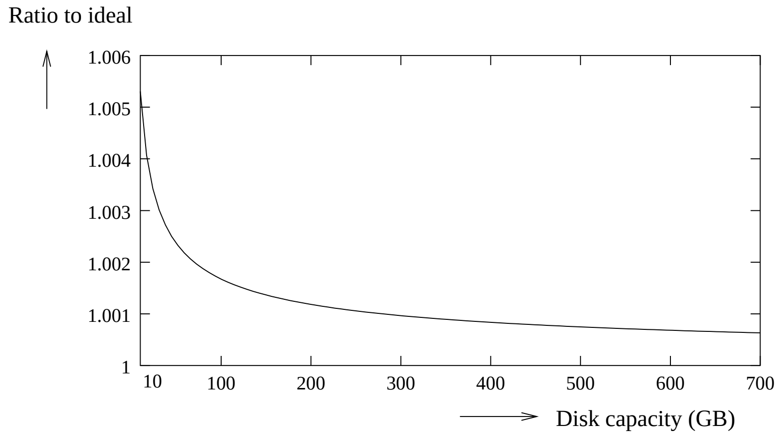


Figure 22: Efficiency on using storage capability (operating on disk)

or more workstations to store huge amount of data and cope with high query arrival rate. Reducing the hardware cost is important and a quantitative method to design the cluster is desired. In the previous work [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12] storage efficiency is not considered and complex simulation is required for performance evaluation. To achieve our research objective, the data allocation algorithm has to allocate variable-size items for both load and storage balancing. The barrier to propose a quantitative method was the lack of such a data allocation algorithm with good ϵ-optimality been proved. The barrier is eliminated in this research. This research changes the design of parallel information retrieval system from ad-hoc approach to systematical and quantitative approach.

References

[1] I. J. Aalbersberg and F. Sijstermans, "High quality and high performance full-text document retrieval: the parallel infoguide system," *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pp. 142–150, 1991.

[2] D. K. Lee, "Massive parallelism on the hybrid text-retrieval machine," *Information Processing and Management, Vol. 31, No. 6*, pp. 815–830, 1995.

[3] C. Stanfill, "Partitioned posting files: a parallel inverted file structure for information retrieval," *Proceedings of the 13th International Conference on Research and Development in Information Retrieval*, pp. 413–428, 1990.

[4] C. Stanfill and B. Kahle, "Parallel free-text search on the connection machine system," *Communications of the ACM, Vol. 29, No. 12*, pp. 1229–1239, 1986.

[5] B. Mansand and C. Stanfill, "An information retrieval test-bed on the cm-5," *Proceedings of the 2nd Text REtrieval Conference*, pp. 117–122, 1993.

[6] B. S. Jeong and E. Omiecinski, "Inverted file partitioning schemes in multiple disk systems," *IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No. 2*, pp. 142–153, 1995.

[7] S. F. Reddaway, "High speed text retrieval from large database on a massively parallel processor," *Information Processing and Management, Vol. 27, No. 4*, pp. 311–316, 1991.

[8] A. Tomasic and H. G. Molina, "Performance of inverted indices in shared-nothing distributed text document information retrieval systems," *IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No. 2*, pp. 142–153, 1995.

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

[10] J. K. Cringean, R. England, G. A. Manson, and P. Willett, "Parallel text searching in serial files using a processor farm," *Proceedings of the 13th ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 429–445, 1990.

[11] S. H. Chung, S. C. Oh, K. R. Ryu, and S. H. Park, "Parallel information retrieval on a distributed memory multiprocessor system," *Proceedings of the 3rd International Conference on Algorithms and Architectures for Parallel Processing*, pp. 163–176, 1997.

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

[13] D. K. Hardman, *Proceedings of TREC Text Retrieval Conference.* 1992.

[14] W. B. Frakes and R. Baeza-Yates, *Information Retrieval: Data Structures and Algorithms.* 1992.

[15] I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes: Compressing and Indexing on Documents and Images.* 1999.

[16] G. Salton, *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer.* 1989.

[17] G. Zipf *Human Behavior and the Principle of Least Effort*, 1949.

[18] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM Journal on Computing, Vol. 3, No. 4*, pp. 299–325, 1974.

[19] W. Dowdy and D. Foster, "Comparative models of the file assignment problem," *ACM Computing Surveys, Vol. 14, No. 2*, pp. 287–313, 1982.

[20] B. Wah, "File placement on distributed computer systems," *Computer, Vol. 17, No. 1*, pp. 23–32, 1984.

[21] J. Wolf and K. Pattipati, "A file assignment problem model for extended local area network environments," *Proceedings of 10th International Conference on Distributed Computing Systems, Vol. 17, No. 1*, 1990.

[22] D. Rotem, G. Schloss, and A. Segev, "Data allocation of multi-disk databases," *IEEE Trans. Knowledge and Data Engineering, Vol. 5, No. 5*, pp. 882–877, 1993.

[23] H. Lee and T. Park, "Allocating data and workload among multiple servers in a local area network," *Information Systems, Vol. 20, No. 3*, 1995.

[24] T. D. C. Little and D. Venkatesh, "Popularity-based assignment of movies to storage devices and video-on-demand system," *ACM/Springer Multimedia System, Vol. 2, No. 6*, pp. 280–287, 1995.

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

[26] L. W. Lee, P. Scheuermann, and R. Vingralek, "File assignment in parallel i/o systems with minimal variance of service time," *IEEE Transactions on Computers, Vol. 49, No. 2*, pp. 127–140, 2000.

[27] D. N. Serpanos, L. Georgiadis, and T. Bouloutas, "MMPacking: a load and storage balancing algorithm for distributed multimedia servers," *IEEE Trans. Circuits and Systems for Video Technology, Vol. 8, No. 1*, pp. 13–17, 1998.

[28] E. Horowitz, S. Sahni, and B. Rajasekaran, *Computer Algorithms/C++*. 1996.

[29] V. P. Kumar, *Introduction to Parallel Computing: Design and Analysis of Algorithms*. 1994.

[30] A. Arpaci-Dusseau, R. Arpaci-Dusseau, D. E. Culler, J. M. Hellerstein, and D. Patterson, "High performance sorting on networks of workstations," *Proceedings of 1997 ACM SIGMOD Conference*, 1997.

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

[32] A. Moffat and J. Zobel, "Self-indexing inverted files for fast text retrieval," *ACM Transactions on Information Systems, Vol. 14, No. 4*, pp. 349–379, 1996.

[33] "Google search engine (http://www.google.com)."

[34] G. Salton and C. Buckley, "Parallel text search methods," *Communications of the ACM, Vol. 31, No. 2*, pp. 202–215, 1988.

[35] H. S. Stone, "Parallel querying of large databases: a case study," *IEEE Computer*, pp. 11–21, 1987.

[36] J. Zobel, A. Moffat, and K. Ramamohanarao, "Inverted files versus signature files for text indexing," *ACM Transactions on Database Systems, Vol. 23, No. 4*, pp. 453–490, 1998.

[37] C. Faloutsos, "Access methods for text," *ACM Computing Surveys, Vol. 17, No. 1*, pp. 49–74, 1985.

[38] A. N. Vo and A. Moffat, "Compressed inverted file with reduced decoding overheads," *Proceedings of the 21st ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 290–297, 1998.

[39] J. Hennessy and D. A. Patterson, *Computer Architecture: a Quantitative Approach, 2nd edition.* 1996.

[40] Y. C. Ma and C. P. Chung, "A dominance relation enhanced branch-and-bound task allocation," *To appear in Journal of Systems and Software.*

[41] C. C. Shen and W. H. Tsai, "A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion," *IEEE Transactions on Computers, Vol. C-34, No. 3*, pp. 197–203, 1985.

[42] M. R. Garey and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness.* 1979.

[43] A. Moffat and L. Stuiver, "Exploiting clustering in inverted file compression," *Proceedings of 1996 Data Compression Conference*, pp. 82–91, 1996.

[44] E. G. Coffman, M. R. Garey, and D. S. Johnson, "An application of bin-packing to multiprocessor scheduling," *SIAM Journal on Computing, Vol. 7, No. 1*, pp. 1–17, 1978.